WHOI-92-11

Software Tools for Acoustic Database Management

by

Kurt M. Fristrup, Mary Ann Daher, Terrance J. Howald
and William A. Watkins

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

January 1992

Technical Report

Funding was provided by the Office of Naval Research
through the Ocean Acoustics Program (code 11250) Contract N00014-88-K-0273
and Grant N00014-J-1445 with supplemental support from NOARL (code 211).

Approved for Distribution:

Peter H. Wiebe, Chairman
Department of Biology

Abstract

Digital archiving of bioacoustic data provides both curatorial and scientific benefits. To realize these benefits, key system requirements must be satisfied. This report discusses these requirements, and describes the software tools developed by the WHOI bioacoustic laboratory to maintain and utilize an archive of digitized biological sounds. These tools are written in standard C code, and are designed to run on PC-compatible microcomputers. Both the usage and structure of these programs are described in relation to the SOUND database of marine animal sounds. These tools include software for analog-to-digital conversion, text header maintenance, data verification and interactive spectrographic review. Source code listings are supplied.

Contents

1 Introduction

Historically, bioacoustics laboratories have maintained libraries of field recordings and prints of spectrographic analyses. Well-organized catalogs for such collections permit routine retrieval of tapes or spectrograms in the inventory. With computer systems, digital technologies can be incrementally added to these collections. A more comprehensive review of potential digital applications suggests that fundamental reorganizations of bioacoustic programs are appropriate.

Bioaccustic studies are increasingly concerned with objective, quantitative analysis of sounds. Such results can be difficult to obtain using spectrographic prints. Many research topics require analysis of thousands of sounds; this would be virtually impossible without dramatic changes in techniques.

Many laboratories are now confronting the difficulty of maintaining or accurately reproducing aging analog tapes in their archives. With analog recording technologies, gradual degradation of the data cannot be avoided. Aging magnetic tape media and tape duplication introduce artifacts and corrupt the original recordings.

Digital technologies provide an immediate solution to the archiving problem. Entire tapes can be converted to digital format and stored on a variety of media. Although no medium is eternal, some optical storage technologies are more stable than magnetic tape. In addition, direct digital copies can be made with extremely low error rates, and such errors can be detected and removed.

Important benefits can be realized even if the entire inventory cannot be digitized. Sound sequences can be digitized and stored as they are analyzed

in the course of research. Re-analysis can proceed with the digitized copy. This minimizes the handling of analog tapes, reducing the risk of damage and degradation from repeated usage. Also, it should be much easier to retrieve an existing series of digitized sounds than to locate the tapes and queue the sounds for re-analysis. Digitized sound sequences also provide "voucher specimens" of the exact acoustic data used to reach scientific conclusions.

The accumulation of digitized sound sequences can provide new opportunities, by increasing the scope of research. For example, digitized sequences produced in the course of studying individual or species-specific repertoires can be combined later to examine the structure and function of these sounds in broader systematic or geographic contexts. To realize these opportunities, a laboratory must coordinate computer resources and solve a variety of data management problems. The value of thousands of digitized sounds is dramatically increased by a flexible means of organizing, selecting, and retrieving them based on associated biological and environmental data (e. g. species, geographic location, season, social context, individual identity, sex and age). We have achieved this capability with the SOUND databases and associated software tools (Watkins, Fristrup and Daher 1991). This report describes the philosophy and implementation of our developing a tool set for bioacoustic research.

2 System Overview

Our basic resource is a large analog tape library accumulated over more than 40 years. The reorganization of these recordings began with the development of a database to improve access to the materials. Each tape is now described in an entry in the SOUND database (Watkins, Fristrup and Daher 1991), which includes the geographic location, time, date, species present, social context, behavioral observations, and other relevant information. SOUND is currently maintained with INMAGIC software (Cambridge, MA). The structure of SOUND makes it a convenient starting point for new research projects. Database queries based on specific research needs can provide an immediate assessment of the available resources, as well as easing their retrieval. We can evaluate the research potential of our library for particular historical periods, geographic regions, phylogenetic groups, types of sounds, etc.

Given the capacity to select tapes of interest, the next need is a means of converting the acoustic data into digital form suitable for computer data processing. Two distinct applications can be identified. An investigator may need to browse tapes to extract particular sounds of interest. This requires a system that is continuously digitizing, and that can be interrupted by the operator to save the most recent portion of the signal. Alternatively, an investigator may wish to convert entire tape recordings into digital form. This latter application requires software that manages the rapid transfer of digital data from an analog-to-digital converter to a large mass storage device.

Before digitized acoustic data are accumulated and software tools are

developed, standard file formats need to be defined. The file format should incorporate a text field containing enough information to ensure proper identification even if external references are lost. The name and storage location of a file can be used to identify sound cuts, but the embedded information is more secure. Digitized sound sequences – or sound cuts – are of little use without a convenient means of searching for and retrieving data of interest. A second database is an obvious means of organizing sound cut information, and it can be closely related to the database catalog of tapes. Our sound cut database is called SOUNDC (Watkins, Fristrup and Daher 1991).

Other requirements for maintaining collections of digital sound cuts are screening the acoustic data for errors in digitizing, cross-checking the text data embedded in each sound cut file against that file's record in the sound cut database, and automatic routines that manage the transfer of validated data to archival mass storage. These are tedious and time-consuming operations; efficient and reliable methods are critical.

A program for reviewing stored sound cuts and manipulating the data for further analysis is required. This should provide visual and aural presentations of the sounds as well as the ability to make simple measurements of signal features. Additionally, the program should be capable of exporting all or portions of a sound cut in convenient file formats for other analyses.

A coordinated system that meets all of these needs is described below. Each function, along with the structure and operation of the relevant software tools, is discussed.

3 Digital File Format

It is important that a bioacoustic collection standardize the format of their digitized sound files. To assure compatibility, we have adopted a header and file format based on the capabilities of a standard commercial analyzer, the Kay Elemetrics 5500 Digital Signal Processor (Pine Brook, N. J.). Their "5500" (KAY) format is used. It has a header of 512 bytes; the data follow in the form of 16 bit binary integers stored in 2's complement, low byte, high byte format.

The header is a mix of binary and ASCII data. The binary data, written by the Kay digitizing equipment, record sampling rate, number of bits of precision, and size of the sound cut. The text field is used for storage of arbitrary notes and information. In the header, we record a portion of the record from the SOUNDC database that describes the data to unambiguously identify the source of the material and the digitizing process.

In the following description of the KAY file header, the bytes in the header are numbered from 1 to 512. Not all fields are listed.

25-26: ASCII '1' '2' -- number of bits/sample.

27- : ASCII, null-terminated string -- number of samples in the file.

65-71: ASCII, "5500SD" -- Kay 5500 "trademark".

121-122: binary integer -- exponent (base 10) for sample rate.

123-124: binary integer -- mantissa for sample rate.

151-512: ASCII text: we insert SOUNDC record information

File names code the identity of the sound cut. The first two characters are the last two digits of the year the recording was made. The next three

characters are a tape number within the year; tapes are usually numbered sequentially as they are cataloged. The last three characters are the number of the digitized sound cut. Because more than 999 cuts may be extracted from a single tape, we use a 36-base numbering system for the cut number. This provides over 40000 unique cut names per tape. The digits are $0-9$, followed by $A-Z$. Thus, the cut after $00Z$ is 010. The remainder of the file name (the MS-DOS file "type") is .KAY. Thus, 8721101A.KAY is the 46^{th} cut (#01A) from the 211^{th} tape recorded in 1987. The first eight characters of the name also provide the unique identifier for the corresponding database record in SOUNDC.

4 Analog-to-Digital Conversion

We satisfy the digitizing requirements of our research with two modes of analog-to-digital conversion. In the interactive mode, data are collected continuously into a ring buffer of random access memory (RAM), with the oldest data being overwritten by fresh samples as they are acquired. This cycling is interrupted by the operator when a signal of interest is complete, and the contents of the ring buffer (the digitized signal) are saved on a mass-storage device as a data file. In practice, we use real-time sound spectrographic processors for interactive data collection: the combination of aural and visual review of taped material promotes more effective signal review and identification. In addition, the interactive mode typically supports higher sampling rates because the data are stored in system RAM, which has much faster storage times. About a minute of sound can be stored at the highest sampling rates in our systems.

In the batch mode, an extended portion of a recording is continuously digitized into a large file, usually on a hard disk. These long data files are subsequently edited or processed automatically to extract signals of interest. Batch digitizing permits larger volumes of data to be acquired with reduced operator involvement. We routinely digitize tens of minutes of continuous sounds at a sampling rate of 80 kHz. Automatic techniques for detecting and characterizing signals are planned to further expand our data processing capacity (Fristrup and Watkins 1992, K. Christian pers. comm., J. Buck pers. comm.).

The Kay Elemetrics 5500 Digital Signal Processor (Pine Brook, N. J.) is a commercial interactive sound spectrograph used for displaying, selecting,

and digitizing sound cuts. The Kay can sample and display one or two channels. Its ring buffer has 8 megabytes of memory, which represents 51.2 seconds of data for a single channel at the highest sampling rate available (81960 Hz). Optional hardware and software permit portions of the data buffer to be downloaded into an ISA bus computer running MS-DOS. The file management software supplied by Kay places binary information (sample rate, sample precision (bits), data length) in a header that precedes the data. The operator can choose to insert notes into this header prior to downloading the signal data.

VOICE is a sound spectrograph developed at the Woods Hole Oceanographic Institution (Martin, Catipovic, Fristrup and Tyack 1990). VOICE consists of an ISA bus microcomputer running MS-DOS, augmented by analog interface and signal processing boards. Its ring buffer is limited to 320 kilobytes of memory, and the maximum sampling rate for gap-free data is about 30 kHz. At the maximum rate, the buffer holds the most recent five seconds of signal. Any portion of the ring buffer can be saved as a signal file. At present, VOICE does not support placing text in the header, but this function is supplied by other utilities described in Section 5.

CSTRM is a batch digitizing program that runs on an AT-compatible microcomputer augmented with analog interface hardware. At present, it is configured to work with the Canetics PC-DMA12 analog interface board. Key features of the PC-DMA12 are DMA circuitry for both A/D and D/A, 100 kHz maximum sampling rate, anti-alias filtering and programmable input amplifiers. CSTRM streams data to hard disk in the KAY format at sampling rates up to 83 kHz.

4.1 CSTRM: A Batch Digitizing System

By: Kurt Fristrup

System requirements: MS-DOS, ISA bus, Canetics PC-DMA12, or an equivalent A/D board (with modifications to the code), a hard disk system (fast and large capacity preferable).

Usage: cstrm <dest> <# 64kbyte blocks> <sample rate in Hz>

Before executing CSTRM, the operator should use a utility that eliminates disk fragmentation. This will minimize seek time required by the hard disk, and permit maximum throughput. The operator also must ensure that there is enough room for the samples requested. At present, CSTRM does not verify that enough disk space is available. We have found it convenient to dedicate a physical disk or logical partition for streamed data and move the data to another storage location after each session to clear the partition for subsequent use.

The command line arguments are the disk/directory destination (a standard DOS path description), the number of 64 kilobyte blocks (32 kilosamples), and the sample rate in Hz. We have digitized up to 190 megabytes of continuous sounds with CSTRM.

Code Description.

CSTRM utilizes subroutines adapted from those supplied by Canetics with their PC-DMA12 analog interface board. We extensively modified the function managing continuous acquisition and transfer to disk. This program is compiled using the COMPACT model of memory management (NEAR code branches, FAR data pointers) to specify memory allocation

functions that use far pointers. This is the only program we have developed that does not use the SMALL model (NEAR code branches and data pointers).

The first statement in main() calls memreq() to obtain a full 64kb page of memory, to serve as the buffer for the DMA data transfers. This function allocates 128kb of memory, and returns a pointer to the first byte (offset=0x0000) on the full page of memory embedded in this block. This is a bit wasteful of memory, but it does not limit the program's function.

The block of conditionals involving argc and argv[ ] access and test the command line arguments. The assignment statements referencing sets→ initialize a data structure used by the routines controlling the analog interface board (AIB). The call to init_board() initializes control parameters in the AIB, and kill_timer() disables the AIB's timer device. This ensures that the board is quiescent prior to initiating data acquisition.

The subroutine kfrdfile_cont() performs the data acquisition. The initial section of the subroutine initializes several pointers that monitor DMA status, shift the data to 5500 format, and transfer the data to disk. The skeletal 5500 header is created and written before data acquisition is initiated.

The general strategy involves dedicating a 64 kilobyte page of memory as a DMA memory buffer, and setting the DMA controller to cyclically transfer data into this page. Thus, the DMA controller resets itself automatically when a full page of data has been acquired, and resumes transferring at the beginning of the page. The program follows the DMA controller's progress, transferring segments of data to disk when they are complete. When the page is divided into two segments, this strategy simplifies to a ping-pong

buffer system. We use 8 segments at present, which leaves 7/8ths of the buffer available to buffer data acquisition against transient delays during disk writes.

The logic of the acquisition/write loop is relatively simple. The high byte of the DMA pointer register is polled repeatedly, and all of the new data (up to the DMA pointer) are shifted right four bits to translate from Canetics integer format (most significant 12 bits used) to 5500 integer format (least significant 12 bits used). If the DMA pointer has passed the end of the current data segment, then these data are written to disk, and the pointers marking the current segment are updated to the next segment in the buffer. The last segment in the buffer is treated differently, because the DMA pointer register rolls back to zero when it increments past the last byte on the page. This mandates a slightly different test to determine when the last segment is full. This code also updates the page count variable, and terminates sampling when the page count equals the requested sample size.

Program termination is signalled by a brief sequence of audible tones generated by the computer's speaker. The program also prints a reminder that the DOS clock may need to be reset, as the timer tick interrupt is disabled during data acquisition.

A slightly faster, multichannel version of this program (CSCRM) has also been developed for acoustic localization. This version does not produce a KAY formatted file: data are streamed to disk in the native format of the (Canetics) PC-DMA12. This uses the most significant 12 bits of a 16 bit word to store each sample, while the KAY format uses the least significant 12 bits.

For multichannel recordings, the analog interface board interleaves data from the different channels into a single file. Accordingly, an additional program (DUMSPLIT) was developed to demultiplex the multichannel information in the composite file and to produce separate Kay files for each channel. In addition to separating the data, DUMSPLIT shifts the sampled values 4 bits to the right to convert from Canetics to Kay sample format.

5 Text Header Maintenance

The text portion of the file header identifies the source of the data and specifies the digitizing settings. This information should be complete enough to replicate the sound cut by re-digitizing the original recording. Headers need to be added to files made with digitizing instruments that do not place text into the header when digitized sounds are saved. Furthermore, changes in database structure or digitizing procedures may require altering the format of the text already in place for a large number of files. Therefore, a program that permits interactive modification of individual headers is needed, along with a second program that processes large groups of headers automatically.

HEADEDIT provides the interactive, file-by-file review and replacement of the header's text. MASSEDIT updates the headers of many files automatically, using the information from a formatted text file that can be imported to, or exported from, the SOUNDC database (Watkins, Fristrup and Daher 1991, pp. 31-32).

5.1 HEADEDIT: Interactive Header Manipulation

By: Kurt Fristrup

System requirements: MS-DOS

Usage: headedit <filename>

HEADEDIT reads the header on the file, and displays three binary fields (sample rate, number of samples, number of bits per sample) and all of the text information. If the two letter codes for fields used in the text database

are recognized in the header, then the text is formatted to display each field separately. Otherwise, the text is displayed unformatted. If the information is correct, the file may be left unchanged.

If the text portion of the header is to be changed, type [Ctrl-n]. A prompt appears for the new text, which may be typed or "pasted" in using a program such as SIDEKICK PLUS (Borland International, Scotts Valley, CA.). When the text is complete, an [Esc] terminates the entry. HEADEDIT ignores more than the first 362 characters entered into the header.

The new header can then be accepted [Ctrl-y], retyped [Ctrl-n], or you can exit the program leaving the header unchanged [Esc]. Our use of Ctrl-y and Ctrl-n for yes or no stems from problems with pasting text using SIDEKICK PLUS: when we attempted to paste too much text into the header, a surplus character was sometimes interpreted as a reply to the prompt. If you type [Ctrl-y], the new header is inserted in the data file, and the program exits. If you type [Ctrl-n], the program cycles back and you are prompted for a new header. [Esc] aborts the program without changing the data file.

Code Description.

HEADEDIT opens the file named in the command line, and reads the first 512 bytes into a buffer. Integer pointers extract the mantissa and exponent of the sampling rate from binary fields in the header. The number of bits per sample and the number of data samples are stored as ASCII text and are extracted using character pointers. The text portion of the header (starting at byte 150) is searched for each of the two letter codes used in the SOUNDC database (e.g. RN). A pointer is assigned to reference each code

found in the header. If the RN (Record Number) field code is not found, the text is displayed verbatim. Otherwise, each field associated with a two letter code is displayed on a new line.

The input section takes text (from the keyboard or a text block from SIDEKICK PLUS) and places it into the header buffer. The character combination that marks the end of a line (carriage return, line feed) is translated to an underscore. This method marks the end of each line, yet allows most software packages to treat the text information as one continuous string.

The program terminates by saving the new header information, [Ctrl-y] or retaining the old header, [ESC].

5.2 MASSEDIT: Batch Substitution of File Headers

By: Kurt Fristrup

System requirements: MS-DOS

Usage: massedit <textfilename> <disk:path\>

With text and data stored in distinct locations, and many researchers contributing to the database, verifying the correspondence of text and data is a significant concern. Additionally, the format and contents of the text database may evolve with time, such that the information in the header becomes out of date. To address these concerns, we use the same text file to update both SOUNDC and the sound cut headers; alternatively, we update the sound cut headers using a text file written from SOUNDC. Thus, MASSEDIT accepts the SOUNDC formatted text records and inserts portions of each record into the corresponding sound cut file.

To prepare for MASSEDIT, a text file must be created with the particular format expected by the SOUNDC database program. In this format, fields within a record begin on a new line, starting with a two letter code that identifies the name of the field. Records are separated by a dollar sign symbol ($) placed on a line by itself.

The first command line argument to MASSEDIT is the SOUNDC text file name. The second, optional argument allows the operator to specify the storage location of the KAY files. If no location is specified, the default directory is used. As the program executes, screen messages indicate flawed text records or failure to locate a sound cut file corresponding to a text record.

Code Description.

MASSEDIT begins by retrieving the text file name from the command line argument list, and opens the file to prepare for reading records. MASSEDIT reads the text for a single record into a buffer (stopping on encountering an end of record dollar sign ($). The buffer is then searched for the letters RN (Record Number), and the eight character number is read. If the RN field is not successfully retrieved, an error is reported, and the first fifty characters of the record are displayed. The program proceeds to the next record without taking further action.

The eight character RN code, with the letters ".KAY" appended, corresponds to the name of the digital sound cut file referred to by the record. MASSEDIT prepends the **disk:path** information to the file name, and attempts to open the file. If MASSEDIT doesn't find, or cannot open the file, an error is reported. This flags the presence of a text record for which

no sound cut was found (perhaps from accidental erasure of the sound cut).

If the file is found, MASSEDIT inserts as much of the SOUNDC text as the header will accommodate (362 characters). End of line characters are translated into underscore characters, to avoid a conflict in the KAY file management software. If the record contains fewer than 362 characters, the remainder of the text buffer is filled with spaces.

The program terminates when the last record has been read and it reaches the end of the text file.

6 Data Verification

Duplication of text and data files can involve large numbers of files and enormous volumes of data. Often, these files may exist on different machines, perhaps even machines that use different operating systems. To verify that all copies are identical, we have devised an automatic procedure based on two public domain utilities: CRC and DIF. Versions of these programs are available for many different machines. This procedure can be applied to any subset of the database, and on different machines in widely separated locations. We present here the sequence of instructions that is appropriate on MS-DOS machines; a similar sequence exists for most other machines.

```
for %c in (*.kay) do crc %c >> dirname.crc
sort <dirname.crc >dirname.srt
```

now copy the .srt files to a common machine, and execute

```
dif dirname1.srt dirname2.srt
```

The initial CRC command generates a single line in DIRNAME.CRC for each filename that satisfies the wildcard specification. In addition to the name and size of the file, a 32 bit CRC value (a common error checking code) is generated for each file. The sort command ensures that all file entries are in alphabetical order, and the DIF command will identify and print differences between the sorted files. This will find file omissions as well as virtually all forms of file corruption.

MASSEDIT will identify text records that have no corresponding sound cut file, and update the headers of all files that do match. It does not

detect sound cut files without a corresponding text record. It also does not check the validity of the digital data. To address these needs we developed KAYCHECK.

KAYCHECK scans the data to ensure that the signal was not clipped during digitizing. KAYCHECK also verifies the presence of a SOUND formatted text field, and checks some of the text values for consistency with the binary data in the header. KAYCHECK provides these checks on every KAY file in a specified directory, and it generates three MS-DOS batch files to assist in subsequent diagnosis of flawed files transfer of validated files, and removal of archival files from the working mass storage. On exiting, KAYCHECK displays the total number of files processed and the number of files with flaws. The batch files automate tedious, time-consuming tasks, and remove the possibility of faulty copy or delete operations due to errors in manually issuing these commands.

6.1 KAYCHECK: Header and Data Verification

By: Kurt Fristrup

System requirements: MS-DOS.

Usage: kaycheck <diskdirectory> [c]

The logical structure of KAYCHECK is relatively simple. Text output files (KCOPY.BAT, KDELETE.BAT, KDIAGNOS.BAT) are opened to receive commands for copying files that passed and diagnosing files that have faults.

Each KAY file (located by Turbo C's findnext() function) is opened and the header is loaded into a buffer. The record number (RN) code is compared with the file name. The sample rate (SR) field is compared with the binary

coded sample rate. The cut size (CS) field is compared with the quotient of the binary coded data length divided by the binary sample rate. If any of these comparisons detect an inconsistency, an error message is displayed on screen and a line is added to the KDIAGNOS file that will run HEADEDIT on this sound cut file to diagnose the problem.

The data in the file are screened for signal clipping that might have occurred during digitizing. At present, clipping is indicated when a binary data value falls below 1 or above 4094 (the maximum A/D sample number). If no evidence for clipping is found, the minimum and maximum sample values are checked to ensure that a sufficient fraction of the sampling dynamic range is used. If less than one-eighth of the dynamic range is used, an underflow message is reported, suggesting that the signal should be redigitized. In either case, a call to SIG (refer below) is inserted into the KDIAGNOS file, and a message is displayed on screen.

If all tests are passed, KAYCHECK creates an entry in the KCOPY batch file that will copy the data file to a new drive and directory and perform a binary file comparison of the copy with the original. KAYCHECK also places an entry in the KDELETE file to assist in the safe recovery of hard disk space after copying.

Code Description.

KAYCHECK begins by forming the wildcard file name using the first command line argument (`drive:directory`) and appending ".KAY". The three batch files (KCOPY, KDELETE, KDIAGNOS) are also opened and assigned to FILE variables. The Turbo C `findfirst()` function is used to locate the first file matching the wild card. These steps complete the

preparation for the principal program loop which follows.

The first statements in the loop increment the file count variable, open the KAY file and determine its length. The header material - the first 512 bytes - are read into a character array. The header material is checked for consistency by the function **headbad()**, which returns a non-zero code if an error is found. The non-zero code causes an appropriate screen message to be displayed and an entry is added to the KDIAGNOS batch file to run HEADEDIT on the KAY file.

If no header errors are found, the data are scanned (**datscan()**) for clipping or underflow (insufficient gain during digitizing). These errors are also indicated by a non-zero code value returned by **datscan()**; an appropriate screen message is displayed and a SIG entry is added to KDIAGNOS.

A counter is used to keep track of the number of files with problems, and when the program terminates, the total number of files scanned and the number with problems are reported.

7 Sound Cut Review

To review stored sound cut files, a flexible tool for browsing, editing and exporting digital acoustic data is required. In addition to providing a graphic representation of the signal, this program provides elementary signal measurement functions and the capability to edit and export acoustic data. The most challenging aspect of developing this utility has been the wide range of signal durations and amplitudes that it must handle. The graphic display should provide a reasonable presentation regardless of the particular structure of the signal.

We have developed a utility that can be used as an independent program (SIG) or in conjunction with the text database (R_SIG, a TSR "pop-up"). R_SIG allows immediate display and manipulation of the digitized signals from within the SOUNDC database program by swapping it out of memory and loading SIG into memory. Both versions provide reasonable spectrographic displays with the default settings, and can work with files of any size. Selected portions of signals can be expanded for more detailed display, or exported to ASCII or MATLAB (The Math Works) formatted files for additional analysis. With appropriate hardware, selected portions of the signal can also be played back for aural review.

7.1 SIG and R_SIG: Sonagram and Waveform Display Utilities

By: Kurt Fristrup and Terrance Howald

System Requirements: MS-DOS, ISA bus, numeric coprocessor, EGA or VGA video graphics system, Canetics PC-DMA12, or (with modifications

to the code) an equivalent D/A board with DMA capability, amplifier and speakers.

Usage: sig filename.KAY [dr] [aa] [e/v]

Usage: tsrint

r_sig <drive:path\ >

R_SIG is subsequently invoked by hitting Alt-Esc.

SIG is a program that produces spectrogram and waveform displays of acoustic data files. Additionally, portions of the data can be selected for more detailed display or exported into ASCII, MATLAB, or KAY format files. R_SIG is a terminate-stay-resident (TSR) program designed to work with the SOUNDC database. When R_SIG is loaded, any file whose record is being reviewed in SOUNDC can be displayed with SIG at the touch of a key. *R_SIG loads a small (10 Kbytes) keyboard monitor into the main memory.* When a hot-key sequence is detected, R_SIG saves the SOUNDC environment to extended memory or hard disk and loads SIG into memory. The KAY file name is taken from the SOUNDC screen by R_SIG, prepended with **drive:path** , and passed to SIG. When SIG is exited, R_SIG restores the SOUNDC environment.

SIG can process long sequences (we have used it on 80 Mb of continuous data). When SIG is invoked, the screen clears and the program begins scanning the data to obtain scaling parameters for the waveform and spectral displays. Just prior to the actual display of the signal spectrogram, time and frequency lines are drawn, forming a grid on the screen. These labelled lines provide a convenient means of interpreting the spectrograph during the drawing of the display, which can take a minute or more for long files.

The grid is erased as the signal spectrogram is drawn. Cursors are used to pick out time and frequency information once the spectrogram display is complete.

The completed waveform and spectrogram displays include basic information regarding the data file. The top line contains the name of the file, the number of samples, the minimum and the maximum A/D conversion numbers, the sample frequency, the dynamic range, analysis attenuator values (the latter two scale the conversion of spectral power to color values) and the FFT size. At this stage, the commands tabled below can be used to position cursors or alter the display parameters. Four cursors are available to bracket features in the waveform and spectrogram display. These provide a means of scaling the image and measuring time or frequency intervals. They can also be used to focus on particular sound sequences by restricting the range of data displayed. This "zoom" function is implemented as a stack with five levels, so successive commands can be used to pick out local details and subsequently return to the more general display.

Other commands include toggling the spectrogram display on/off, toggling noise compensation on/off, adjusting dynamic range and analysis attenuator, saving a section of the data to a new file, listening to the data between the time cursors, changing the FFT size, and exiting the program. The format of an exported data file from SIG is determined by the extension specified for the name (.TEX, .MAT, or other). TEX files are saved in ASCII, MAT files are saved in MATLAB (MathWorks) format, and any other extension results in a KAY format.

COMMANDS FOR SIG

left, right arrow	move a time cursor left, right.
up, down arrow	move a frequency cursor up, down.
ctrl left, right arrow	move a time cursor 5x speed.
Home <keypad>	toggle front/back cursors.
PgDn <keypad>	zoom in to new display bound.
PgUp <keypad>	pop back to previous display bounds.
Alt-P	listen to data between time cursors.
Alt-F1	multiply playback rate by two.
Alt-F2	divide playback rate by two.
Ctrl PrtSc	export data between the cursors
Esc	exit the program!
Alt-F	toggle spectrogram display!
Alt-N	toggle noise compensation!
Alt-D	restore default parameters!
Alt-F5	multiply fft size by two!
Alt-F6	divide fft size by two!
Insert <keypad>	increase dynamic range by 3db!
Delete <keypad>	decrease dynamic range by 3db!
Ctrl-PageUp	increase analysis attenuator by 1db!
Ctrl-PageDn	decrease analysis attenuator by 1db!

! -- these commands are effective during screen draw

Code Description.

The code for SIG is split into several files, which are loosely organized by function. The core subroutines are in COMSIG.C, including **main()**. All routines related to FFT processing are in FFTFUNCS.C. All simple routines related to screen display are in PLOTUTLS.C. WHTIMIO.C contains a few input/output routines related to data access and error reporting. NDRAWVLN.C and DRAWHLN.C are graphics subroutines that utilize inline assembly code to maximize line drawing speed. NDRAWVLN.C is roughly ten times faster than the Turbo C library function.

Main() copies the first command line argument (the KAY file name) into a global string variable **filename**, sets global video constants and spectrogram scaling parameters, and passes control to **pstest()**. This structure reflects our adaptation of the previous generation TSR support routines from South Mountain Software (South Orange, N. J.). This required that for the program to be activated as a TSR, it had to be a subroutine. In that version, there was no **main()** in COMSIG.C, and the TSR subroutine used **pstest()**.

Pstest() switches the video screen to graphics mode, opens the KAY file to read the data, and initializes global variables in the FFTFUNCS.C file using functions provided for that purpose. The header is loaded and the sample rate is decoded from two integer fields. Next, **pstest()** calls **getscale()**, which selectively scans the file and determines appropriate scaling factors for the waveform and spectrogram displays.

Getscale() does not always read every data value in the file. Exceedingly long files are sampled at regular intervals, and the scaling factors are

estimated from these samples. **Getscale()** also estimates the background noise spectrum for the sound cut. The sampled data blocks are ranked by acoustic intensity, and those that fall between the fifth and tenth percentiles are used to form an average noise power spectrum. This spectrum can be used to adjust the spectrogram display such that the background noise is "whitened".

After scaling factors are determined, the screen display commences. **Lineplot()** performs the waveform and spectrogram display. If a keystroke is sensed while **lineplot()** is executing, the routine aborts and returns the keystroke code. This permits the adjustment of the scaling factors without having to wait for the screen display to complete.

When the graphics picture is complete, a brief header and footer are *written to summarize* the salient characteristics of the sound cut. Then, the program waits for a keystroke that would further adjust the scaling parameters, activate time or frequency cursors, zoom in on a selected portion of signal, or export a portion of the signal.

Any keystrokes that change the screen display (changing dynamic range or analysis attenuator, modifying FFT size, etc.) cause **getcurses()** to exit and pass control back to the principal program loop. The exit code is used to indicate which function has been requested. After the proper parameters have been changed, program control cycles back to **lineplot()**.

The organization of the graphics display is constrained by the specifications of the EGA and VGA graphics standards. There are 640 vertical lines in the display, so the portion of the file being displayed is divided into 640 equal segments. If there are fewer than 640 segments of 256 samples

each in the file (163840 samples), the envelope display is completed (no samples skipped during min/max search), and the spectrogram is based on a 256 sample segment. Because the spectrogram buffer includes more data than each waveform segment, the spectrogram will tend to lead the waveform display. In an extreme display of one sample per vertical line, features will appear in the spectrogram display nearly one-half screen ahead of the waveform display.

If there are more than 163840 samples in the file, data will be skipped in computing the waveform and spectrogram displays. The spectrogram display becomes the average of several subsegments within the data spanned by one vertical line, and the waveform is the range of sample values observed in those subsegments.

When noise compensation is enabled, the color scaling on screen is related to the noise floor for each frequency bin, but when this is disabled, all frequency bins are scaled using the same standard. The noise floor is established by averaging 64 power spectra formed from segments that fall between the fifth to tenth percentiles in observed rms values. In practice, noise compensation tends to bring out weak signals at higher frequencies.

The dynamic range parameter adjusts the range of values spanned by the fifteen colors used in the display. The analysis attenuator parameter determines the range of values placed in the same color bin as the loudest signal found. In a linear relationship between intensity (in dB) and color value, the dynamic range sets the slope of that relationship, and the analysis attenuator determines the offset.

The playback feature of SIG utilizes the D/A capabilities of the Canet-

ics PC-DMA12 board, or an equivalent analog interface system. The initial playback rate is set equal to the sample rate. The user may increase or decrease the playback rate from a minimum of 640 Hz up to a maximum of 81920 Hz. D/A playback is initiated by entering [Alt-P] to call the function **start_dac()**. Parameters passed to **start_dac()** are the first and last data points, the sampling rate, and a scaling factor for the sampling rate. This sound playback routine executes in a manner similar to CSTRM, with the exception that **start_dac()** performs D/A conversions instead of A/D conversions. **Start_dac()** was developed from routines distributed by Canetics (Pasadena, CA) with their PC-DMA12 analog interface board.

Start_dac() allocates a 64kb page of memory by calling **memreq()** and sets the data pointer to the first data point. The program then determines the number of samples, the user selected playback rate (sample rate times the scaling factor), and the number of 64kb pages to be converted. Next, the routine initializes the AIB and executes **wtfile_cont()** which controls the D/A process.

Wtfile_cont() examines the amount of sound data to be analyzed. If this amount is less than 65535 (64kb), then the data are read from the hard disk to memory, converted from the KAY integer format to the Canetics integer format, and sent via DMA to the AIB in 16kb blocks. If the last 16kb block is not complete, the remaining portion of the block is filled with the value of the last data point. The PC-DMA12 operation is halted after execution of the last block.

If the amount of data to be converted is greater than 64kb, then the first 64kb of the data is loaded into memory, and the conversion process is started.

After the memory pointer passes the first 16kb, the old data is overwritten by the next 16kb of data in the sound sequence. This process of sending a block to the AIB and then filling that block with new data continues until less than 16kb of remain. The last block is handled as described above, and the function terminates

The code for R_SIG consists of one file. Many of the functions used in **main()** are adapted from those provided by South Mountain Software (Pasadena, CA). The first code in **main()** checks to see if the switch "rel" is included in the command line. If the switch is included, the program attempts to remove R_SIG from memory. The program reports successful removal or failure to find R_SIG in memory. If the switch is not present, the program checks to see if R_SIG is already loaded or if a path to the data files is included in the command line. If there are no problems with the command line, **swap_tsr()** decides where to temporarily store the SOUNDC data. Then **inittsr()** loads a small kernel of 10kb into the main memory.

This kernel monitors the keyboard for a hot-key sequence. When the key sequence [Alt-Esc] is detected, **run_sig()** is executed. This routine gets the name of the KAY file being examined in SOUNDC from the video memory and prepends "SIG" at the beginning of the string and ".KAY" at the end of the string. SOUNDC is then swapped out of memory and SIG filename.KAY is executed. Once SIG is exited, SOUNDC is swapped back into memory and the user can continue from the point where the hot-key sequence was entered.

8 Conclusions

The system that has been developed for curating and analyzing marine animal sounds from our SOUND databases has evolved over several years. These software tools have satisfied our needs remarkably well, and we anticipate that they could be useful to others for similar signal database management. Their functions are summarized as follows:

- Maintenance of a database catalog of field recording resources and their contents.
- Integration of resources for interactive and batch conversion of field recordings to digital form suitable for computer data processing.
- Adoption of a standard file format for digitized sound sequences that includes a text field for identification.
- Maintenance of a database catalog of the digital sound sequences and their contents.
- Creation and revision of text information attached to sound data files.
- Verification of the presence and consistency of text information in sound data files.
- Validation of the sound sequence data (no clipping, sufficient dynamic range).
- Coordination of means for finalizing the animal sound data, copying validated sound files, diagnosing flawed files, and removing files from working mass storage once they have been archived.

- Utilization of tools for reviewing sound data files, making simple measurements, and exporting all or portions of files in convenient formats for additional analysis.

The WHOI marine animal SOUND database system was developed before commercial database programs were available to store and manipulate arbitrary binary data in concert with text. Several programs now offer this capability, including the capacity to develop custom routines for displaying and manipulating the binary data. These programs would remove the necessity of coordinating a database catalog with separate archives of digitized sound sequences. However, they would not offer equivalent latitude in analytical procedures. A critical factor in evaluating such databases is the flexibility allowed for custom programming and the ease of developing such analytical routines.

The evolution of research requirements emphasizes the need for flexibility. The SOUNDC database was designed to encourage selection and retrieval of data files based on biological criteria. Our recent work has focussed on automatic characterization of marine animal sounds (Fristrup and Watkins 1992) requiring analysis and comparison of sounds from many species. This could have been accomplished by performing separate select and copy operations for data associated with each species, with distinct disk directories dedicated to each set of data. However, it was much easier to process all relevant sounds without prior sorting. We subsequently linked all of the numeric results to their respective SOUNDC entries, and used database queries to sort and summarize the results.

We discovered that INMAGIC software currently used by the SOUND databases was not flexible enough to conveniently support this kind of linkage between text information (which remains unchanged) and varying amounts of numeric information. Relational database programs (such as PARADOX, Borland International, Scotts Valley, CA) supply this flexibility by providing linkage between separate tables of information. The SOUNDC data was translated, therefore, into a PARADOX table that carried the biological information. The analytical results were tabulated separately, with a file name and the related numerical values constituting an entry. The SOUNDC record identifier and the file name field in the numerical tables provided the means of linking the tables. Categorical summaries were obtained by performing queries based on the SOUNDC information, and reporting linked numerical data specified by PARADOX.

It is not possible to foresee future research requirements, but extensive retooling can be avoided by sufficient scientific insight and effective data management. The time invested in acquiring and annotating the digital bioacoustic data will be well spent if the resulting system facilitates established protocols and fosters innovative research.

Acknowledgements

These software tools for acoustic database management have evolved to meet the needs of the bioacoustic research community at WHOI. Many key features were inspired by thoughtful suggestions from our colleagues. We particularly thank Peter Tyack and Laela Sayigh for their contributions, and Lisa Taylor for help with the manuscript.

The main source of funding for the marine animal SOUND database system has been the Office of Naval Research through a series of contracts and grants, including N00014-91-J-1445 from the Ocean Acoustics Program. Supplemental support for specific enhancements of the marine animal sound data have been provided by NOARL (Stennis Space Center, MI), and ORINCON/DARPA (NOSC,San Diego). Funding for particular analytic assessments of sound characters has been through TRICCSMA (NUSC, Newport) from NAVSEA, Contract Number N00140-90-D-1979.

Literature Cited

Fristrup, Kurt M. and William A. Watkins 1992. Automatic characterization of marine animal sounds. Technical Report WHOI-92-4, Woods Hole Oceanographic Institution, Woods Hole, MA. 02543, 74pp.

Martin, Ann, Josko A. Catipovic, Kurt Fristrup, and Peter L. Tyack 1990. VOICE - A spectrogram computer display package. Technical Report WHOI-90-22, Woods Hole Oceanographic Institution, Woods Hole, MA. 02543, 95 pp.

Watkins, William A., Kurt Fristrup, and Mary Ann Daher 1991. Marine animal SOUND database. Technical Report WHOI-91-21, Woods Hole Oceanographic Institution, Woods Hole, MA. 02543, 56 pp.

9 Appendix: Source Code Listings

These listings represent "working code"; they are not necessarily free from bugs or structured in the most logical fashion. Continued development and modification are anticipated. Most of the code for these programs was developed using Turbo C++ (Borland International, Scotts Valley, CA); the exceptions are the 87FFT library (Microway, Kingston, MA) and some video graphics functions that were coded in assembly to increase screen drawing rates. We used standard C code to enhance portability (object-oriented extensions were not used). Compiler switches were set – for efficiency – to generate 80286 code, use a numeric coprocessor, and maximize register and jump optimizations.

```
/*
Gutted Canetics routines for fast streaming to disk
MUST BE COMPILED WITH THE COMPACT MODEL!!!!!!
kf 4/26/90
*/
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <conio.h>
#include <fcntl.h>
#include <lib180.h>
#include <stat.h>
#define    CONTROLA     /* add+1*4 */     15876
#define    CONTROLB     /* add+2*4 */     15880
#define    OFFSET_ADD     /* add+10*4 */ 15912
#define    RESET_ADD     /* add+11*4 */ 15916
#define    DMAMODE_ADD     11
#define    DMA_MASK_ADD     10
#define    CLRPTR_ADD     12
#define    AD_TIMER_MODE     122          /* ad is timer #1 in this mode */
#define    DA_TIMER_MODE     52          /* and da is timer #0 */
#define    TIMER_AD          15925
#define    TIMER_DA          15924
#define    TIMER_ADD          15924
#define    TIMER_MODE_ADD 15927
#define    STA     15904
#define    STD     15908
#define    TOGGLE_DAC_ADDRESS 15888
#define    BLOCKSIZE 0x2000
typedef struct settings *psets;
struct settings sets1;
psets sets;
int msb, lsb, array_size;
char     c;
unsigned     num_of_transfers;
char     filename[30], curfilname[30];
int     file1;
int     file2;
unsigned     runstop, mline, gainbits, inmode, offset;
char     fourloadflag, flag;
long sets_tmp[11];
char bits[9] ;
/*
UH_SEG returns the segment portion of an anti-hugified pointer
UH_OFF returns the segment portion of an anti-hugified pointer
*/
#define UH_SEG(p) (( FP_SEG( p ) + ((FP_OFF( p ) ) >> 4) ) & 0xF000)
#define UH_OFF(p)
((int)((((FP_SEG(p)+((FP_OFF(p))>>4))&0x0FFF)<<4)+(FP_OFF(p)&0x000F)))
void soundoff(int frq, long dur){
     long i;
     sound(frq);
     for(i=0;i<dur;i++);
     }
int far *memreq(void){
     int tempseg;
     void far *death_to_nonbelievers ;
     death_to_nonbelievers = farmalloc(0x20000L);
     if (death_to_nonbelievers == NULL){
          printf("Coreleft = %ld     Need %ld\n\r", coreleft(), 0x20000L );
          putchar(1);
          exit(1);
```

```
        }
    tempseg = UH_SEG(death_to_nonbelievers);
    if (UH_OFF(death_to_nonbelievers) != 0)
        tempseg += 0x1000;
    return (int far *)MK_FP(tempseg, 0);
    }
/*
for digital I/O and muxes
*/
#define CONTROL_ADD 15876
#define OUTPORT_ADD 15932
#define STROBE          15892
/*
gutted lib180.c for efficient streamer
kf 4/26/90
*/
/*
This routine initializes the PC's DMA controller prior to doing conversions.
It masks DMA channel 1 on, sets the channel's mode, transfer direction,
starting address, and number of transfers to be made.
*/
void init_dma_ad(unsigned count, unsigned seg){
    outp(DMA_MASK_ADD, 0x05);
    outp(DMAMODE_ADD, 0x45);
    outp(CLRPTR_ADD, 0);
    outp(2, 0);
    outp(2, 0);
    outp(3, count & 0x00FF);
    outp(3, count >> 8);
    outp(130, seg);
    outp(131, seg);
    outp(DMA_MASK_ADD, 1);
    }
/*
This routine initializes the PC's DMA controller prior to doing conversions.
It masks DMA channel 3 on, sets the channel's mode, transfer direction,
starting address, and number of transfers to be made.
*/
void init_dma_da(unsigned count, unsigned seg){
    outp(DMA_MASK_ADD, 0x07);
    outp(DMAMODE_ADD, 0x4B);
    outp(CLRPTR_ADD, 0);
    outp(6, 0);
    outp(6, 0);
    outp(7, count & 0x00FF);
    outp(7, count >> 256);
    outp(130, seg);
    outp(131, seg);
    outp(DMA_MASK_ADD, 3);
    }
void init_dma_ad_cont(unsigned seg){
    outp(DMA_MASK_ADD, 0x05);
    outp(DMAMODE_ADD, 0x55);
    outp(CLRPTR_ADD, 0);
    outp(2, 0);
    outp(2, 0);
    outp(3, 0xFF);
    outp(3, 0xFF);
    outp(130, seg);
    outp(131, seg);
    outp(DMA_MASK_ADD, 0x01);
    }
/*
This int initializes the control registers on the PC-DMA using the
```

```
offset, mline, runstop, datasize, inmode, gainbits, and filterstate
global variables. It also resets the board before and after to insure
that the board's controlling state machine is in the proper state to
start conversions.
*/
void init_board(psets sets){
    unsigned temp;
    temp = sets->mline + (sets->runstop << 5) + (sets->filterstate << 6) ;
    outp(RESET_ADD, 0);
    outp(OFFSET_ADD, sets->offset);
    outp(CONTROLA, temp + (sets->datasize << 7));
    outp(CONTROLB, 2 + (sets->inmode << 2) + (sets->gainbits << 4) + 128);
    outp(RESET_ADD, 0);
    if (sets->datasize == 0){
        outp(CONTROLA, temp + 128);
        outp(CONTROLA, temp );
        }
    if (sets->dac2flag == 2)
        outp(TOGGLE_DAC_ADDRESS, 0);
    } /* init_board() */
/*
This function disables all the counters in the PC-DMA's on board timer
device. Use this routine before changing any of the board's operating
parameters.     The timer activation routine should be the last thing called
before conversions start.
*/
void kill_timer(psets sets){
    outp(RESET_ADD, 0);
    outp(CONTROLB, 15 + 16*sets->gainbits + 128);
    outp(RESET_ADD, 0);
    if (sets->dac2flag == 2)
        outp(TOGGLE_DAC_ADDRESS, 0);
    outp(TIMER_MODE_ADD, 50);
    outp(TIMER_MODE_ADD, 114);
    outp(TIMER_MODE_ADD, 178);
    init_dma_da(0, sets->seg);
    init_dma_ad(0, sets->seg);
    }
/*
This routine initializes the timer device on the PC-DMA, setting it
up to generate trigger pulses spaced (spaces) microseconds apart.
Note that the A/D's timer is operated in a different mode from that
of the D/A.     The D/A's timer is a straight divide by n counter producing
triggering pulses every n microseconds. The A/D's timer, on the other
hand, is triggered by the D/A's timer and functions as a programmable
delay. It is used to produce a triggering pulse n microseconds after
the D/A timer produces a pulse. This was int}ed to allow Input and
Output conversions interleaved at a rate determined by the D/A's timer,
and spaced (D/A to A/D period) by the A/D's timer. In order to use the
A/D alone the programmer sets up the timers for interleaving but sets the
D/A to be triggered by software control, effectively disabling it as far
as the timers are concerned.
*/
/*
This sets up the timers for both the A/D and D/A in interleaved mode.
if only the A/D is being used, set the inmode variable to make the D/A
controlled by software strobe, disabling timer control of it.
*/
void init_timer_ad(unsigned spaces){
    outp(TIMER_MODE_ADD, DA_TIMER_MODE);
    outp(TIMER_MODE_ADD, AD_TIMER_MODE);
    outp(TIMER_DA, spaces & 0x00FF);
    outp(TIMER_DA, spaces >> 8 ) ;
    outp(TIMER_AD, 2);
```

```
    outp(TIMER_AD, 0);
    } /* init_timer_ad() */
/*
sets up the program for streaming to disk
modified by kf 4/26/90
*/
void kfrdfile_cont(psets sets, char *curfilname){
    unsigned jbuff, curfil, ipage, blksz2, blksiz, blksiz1, pofset, lastbuff;
    unsigned far *blkindx, *irotptr, *blkptr;
    kill_timer(sets);
    blksiz1 = (blksiz=BLOCKSIZE) - 1; blksz2 = blksiz>>1;
    lastbuff = (256-(blksiz>>8))<<8;
    curfil = open(curfilname,
                  O_WRONLY|O_CREAT|O_TRUNC|O_BINARY,
                  S_IWRITE|S_IREAD);
    blkptr = MK_FP(sets->seg<<12,0);
    memset(blkptr,0,512);
    *(blkptr + 60) = 2;
    *(blkptr + 61) = 10000/sets->waits;
    write(curfil,blkptr,512); /* reserve space for header */
    sets->inmode = 1;
    init_board(sets);
    init_dma_ad_cont(sets->seg);
    printf("reading...\n");
    outp(0x21,0xb9); /* kills the dos clock interrupt */
    ipage = 0; jbuff = blksiz1; blkindx = (irotptr = blkptr)-blksz2;
    init_timer_ad(sets->waits);
    outp(CLRPTR_ADD, 0);
    while(ipage < sets->num_of_pag){ /* ipage is 64k page # */
        while(jbuff < 0xffff){          /* jbuff=ofs lastbyte in current buffer */
            while(inp(2),jbuff >= (pofset = (unsigned)inp(2)<<8))
                while(FP_OFF(irotptr) < pofset)
                    *irotptr++ >>= 4;
            while(FP_OFF(irotptr) <= jbuff)
                *irotptr++ >>= 4;
/*            printf("%p %p %u\n",irotptr, blkindx, jbuff);*/
            write(curfil,blkindx+=blksz2,blksiz);
            jbuff += blksiz;
            } /* first n-1 buffers: while jbuff < 0xffff */
        while(inp(2),lastbuff <= (pofset = (unsigned)inp(2)<<8))
            while(FP_OFF(irotptr) < pofset)
                *irotptr++ >>= 4;
        while(FP_OFF(irotptr) > 0)
            *irotptr++ >>=4;
/*        printf("%p %p %u\n",irotptr, blkindx, jbuff);*/
        write(curfil,blkindx+=blksz2,blksiz);
        ipage++; jbuff = blksiz1;
        }
    kill_timer(sets);
    outp(0x21,0xb8); /* restores the dos clock interrupt */
    close(curfil);
    } /* read_file_cont() */
void main(int argc, char *argv[]){
    int far *M_Ptr;
    long sloop;
    M_Ptr = memreq();
    sets = &sets1;
    if (argc < 3){
        printf("usage: cstrm <filename> <# 64 kb blocks> <samplerate Hz>\n");
        exit(-1);
        }
    else if (argc < 4)
        sets->waits = 20;
    else{
```

```
    if (0.0==atof(argv[3])){
        printf("problem converting %s to sample rate\n",argv[3]);
        exit(-1);
        }
    sets->waits = 1.0e6/atof(argv[3]);
    if (14 > sets->waits){
        printf("maximum sampling rate for DELL 316sx system is 71428Hz\n");
        exit(-1);
        }
    if (atoi(argv[2]))
        sets->num_of_pag = atoi(argv[2]);
    else{
        printf("problem converting %s to number of 64kb blocks\n",argv[2]);
        exit(-1);
        }
    }
printf("%d microsecond sampling\n%d 64kb blocks\n%e minutes of sampling\n",
        sets->waits,sets->num_of_pag,
        (double)65536.0e-6*sets->num_of_pag*sets->waits/60.0);
sets->datasize = 1;                     /* 12 bit sampling */
sets->mline = 0;                        /* one channel sampling */
sets->runstop = 0;                      /* no multiplexer cycling */
sets->gainbits = 0;                     /* set gain = 10 */
sets->inmode = 3;
sets->offset = 128;
sets->filterstate = 0;
sets->usingxmux     = 0;
sets->dac2flag = 2;
sets->seg = FP_SEG( M_Ptr ) >> 12 ;
num_of_transfers = sets->num_of_con     * 2;
init_board(sets);
kill_timer(sets);
kfrdfile_cont(sets, argv[1]);
for(sloop=400;sloop<1000;sloop=(sloop*109)/100)
    soundoff(sloop,25000);
for(sloop=1000;sloop>400;sloop=(sloop*100)/109)
    soundoff(sloop,25000);
nosound();
printf("you may need to reset the dos clock! use ncc\n");
} /* main */
```

```
/*
HEADEDIT: inspect and edit *.KAY file headers
*/
#define YES 0x19
#define NO 0xe
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
main(argc,argv)
int argc;
char *argv[];
{
    FILE *fin;
    long samprate;
    char ch,*hdbeg, *hptr[40], header[513];
    int icmd, i, *imant, *iexp;
    char *strcode = "RN \0CU \0NC \0SR \0CS \0PL \0SC \0ID \0AG \0IA \0GS \
\0GA \0OD \0NT \0DA \0GB \0GC \0OT \0SH \0OS \0NA \0DI \0BH \0HY \0RC \0RG \
\0RB \0RL \0ST \0SL \0SD \0SP \0FF \0FS \0SB \0FE \0AU \0LO ";
    header[512]=0;
    clrscr();
    fin = fopen(argv[1],"r+b");
    if (argc != 2){
        printf("incorrect command tail\nuse headedit fname\n");
        clrscr();
        exit(2);
        }
if (fin == NULL){
        printf("%s not opened\n",argv[1]);
        clrscr();
        exit(1);
}
    fread(header,1,512,fin);
    imant = (iexp = header + 120) + 1;
    samprate = *imant;
    while(*iexp--)
        samprate = samprate*10;
    icmd = 0;
    hdbeg = header+150;
    printf("%s\n",argv[1]);
    printf("sample rate: %ld\n",samprate);
    printf("bits: %c%c\n",*(header+24),*(header+25));
    printf("samples: %s\n",header+26);
    if(NULL==strstr(hdbeg,"RN ")){
/* dump contents on screen for diagnosis */
        for(i=0; i<512-150; i++)
            if(i%70)
                printf("%c",*(hdbeg+i));
            else
                printf("%c",*(hdbeg+i));
        } /* RN not found */
    else{ /* RN found */
        do{
/* tag all recognized inmagic codes with pointers*/
            hptr[icmd] = strstr(hdbeg,strcode);
            if ((hptr[icmd] != hdbeg)&&(hptr[icmd]!=NULL))
                *(hptr[icmd]-1) = 0x7f;
            strcode += 4;
            icmd++;
            }while(*strcode&&(icmd<40));
        for(i=0;i<512-150;i++)
            switch (*(hdbeg+i)){
/* place nulls before all inmagic codes found */
```

```
                    case 0x7f :
                        *(hdbeg+i)=0;
                        break;
/* convert end of lines to underscores */
                    case 0x0a :
                    case 0x0d :
                        *(hdbeg+i)='_';
                        break;
                    }
            }
/* print out inmagic fields */
            for(i=0;i<icmd;i++)
                if(NULL!=hptr[i]){
                    printf("%s\n",hptr[i]);
                    }
        /* if change desired, read from stdin and write into the file */
        printf("Is the header correct (ctrl-Y, ctrl-N, or ESC)?");
            do{
                ch=toupper(0xff&getch());
                if(27==ch)
                    clrscr(),exit(0);
                }while(YES!=ch && NO!=ch);
            while(NO==ch){
/* ok, we want to change the header information */
                clrscr();
                printf("paste using SideKick now, end with <Esc>\n");
                for(i=0;i<512-150;i++){
                    if(0x1b==(ch=getch()&0xff))
                        break;
                    if((0xa==ch)||(0xd==ch))
                        ch='_';
                    putch(ch);
                    *(hdbeg+i)=ch;
                    }
                for(;i<512-150;*(hdbeg+i++)=' ');
                clrscr();
                for(i=0;i<512-150;i++){
                    putch(*(hdbeg+i));
                    }
                printf("Is this correct? ");
                do{
                    ch=0xff&getch();
                    if(27==ch)
                        clrscr(),exit(0);
                    }while(YES!=ch && NO!=ch);
                if(NO==ch)
                    continue;
                fseek(fin,(long)150,SEEK_SET);
                for(i=0;i<512-150;fputc(*(hdbeg+i++),fin));
                } /* while NO == ch */
        fclose(fin);
        clrscr();
        return(0);
        } /* main */
```

```
/*
MASSEDIT: automatically update .KAY files using inmagic .TXT files
for every RN in the .TXT file, look for a corresponding .KAY file
and insert the appropriate header information
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TSIZE 512-150
main(argc,argv)
int argc;
char *argv[];
{
    FILE *fin, *fout;
    int j, i;
    char ch, headtext[TSIZE], ofname[80];
    if (argc < 2){
        printf("incorrect command tail\nuse massedit fname\n");
        exit(2);
        }
    if (NULL == (fin = fopen(argv[1],"r"))){
        printf("%s not opened\n",argv[1]);
exit(1);
        }
    do{
        for(i=0;i<TSIZE && !feof(fin);i++)
            if('\n'==(ch=getc(fin)))
                headtext[i]='-';
            else if('$'==ch)
                break;
            else
                headtext[i]=ch;
        if(TSIZE==i) /* clear the rest of the record */
            while('$'!=getc(fin)&&!feof(fin));
        else
            for(j=i;j<TSIZE;headtext[j++]=' ');
        while('\n'!=getc(fin) && !feof(fin)); /* clear the newline character */
        strcpy(ofname,argv[2]);
        j=sscanf(headtext," RN %[0-9A-Za-z]",ofname+strlen(ofname));
        if(0==j){
            printf(">%55.50s< not scanned\n",headtext);
            continue;
            }
        strcat(ofname,".kay");
/*          for(j=0;j<TSIZE;printf("%c",headtext[j++]));
        printf(" <-\n");*/
        if(NULL==(fout=fopen(ofname,"r+"))){
            printf("error finding %s\n",ofname);
            continue;
            }
        fseek(fout,(long)150,SEEK_SET);
        fwrite(headtext,1,TSIZE,fout);
        if(0!=fclose(fout))
            printf("error closing %s\n",ofname);;
        memset(headtext,0,TSIZE);
        } while(!feof(fin));
    fclose(fin);
    return(0);
    } /* main */
```

```
/*
KAYCHECK
check all .KAY files for proper annotation
displays each file that fails the test
*/
#define HILIMIT 4095
#define LOLIMIT 0 /* limits for 12 bit integer sampling in KAY format */
#define HEADBYTES 512
#define NOTESTART 150
#define RNNO 1
#define SRNO 2
#define CSNO 3
#define RNBAD -1
#define SRBAD -2
#define CSBAD -3
#define SIZBAD -4
#define BUFFSIZE 4096
#define BUFFSIZE2 8192
#include <stdio.h>
#include <io.h>
#include <dir.h>
#include <fcntl.h>
#include <time.h>
#include <math.h>
#include <string.h>
unsigned RN = ((unsigned)'N'<<8)|(unsigned)'R';
unsigned SR = ((unsigned)'R'<<8)|(unsigned)'S';
unsigned CS = ((unsigned)'S'<<8)|(unsigned)'C';
/*
ERREXIT
*/
void errexit(char *msg){
     fcloseall();
     printf("error: %s\n",msg);
     getch();
     exit(-1);
     }
/*
FOPENINP
*/
long fopeninp(char *fname, int *ifile){
     *ifile = open(fname,O_RDONLY|O_BINARY);
     while (-1 == *ifile){
          printf("Source File opening did not succeed: %s\n",fname);
          printf("Enter file name for input file: ");
          if (!scanf("%s\n",fname))
               errexit("aborted from file entry");
          *ifile = open(fname,O_RDONLY|O_BINARY);
          }
     lseek(*ifile,0,SEEK_END);
     return(tell(*ifile)); /* return number of bytes */
     }
/*
DATSCAN
reads integer data from binary ifl, checks for overrange errors
computes summary statistics and writes these to text file sfl
*/
int datscan(int ifl, FILE *sfl){
     int ii, ibuff[BUFFSIZE], *iptr, ired, ival, imin, imax;
     imin=imax=(HILIMIT-LOLIMIT)/2;
     while(!eof(ifl)){/* MAIN DATA LOOP */
          ired = read(ifl,ibuff,BUFFSIZE2)>>1;
          for(ii=0, iptr=ibuff; ii++ < ired; ){
               if(LOLIMIT >= (ival=*iptr++))
```

```
                return(-1);
            else if (HILIMIT <= ival)
                return(-1);
            if(imin>ival)
                imin=ival;
            else if (imax<ival)
                imax=ival;
            }
        }
    if((imax-imin)*(long)16<(long)(HILIMIT-LOLIMIT))
        return(-2); /* underflow */
    else
        return(0);
}
/*
FINDLBL
looks for a two character sequence in a null terminated string
*/
char *findlbl(char *buff, unsigned key){
    unsigned *iptr;
    while(iptr=buff,*buff++ && *iptr!=key); /* move int ptr bytewise */
    return((*--buff? buff:NULL));
    }
/*
HEADBAD
checks for inconsistent recno, sample rate, variant cut size
returns an int error id
*/
int headbad(char *fname, char *hdr, long fsiz){
    double cutsize;
    long samprate, hsrate, nsamples;
    int *imant, *iexp;
    char *keyptr, sfield[80];
    imant = (iexp = hdr + 120);
    samprate = *++imant;
    sscanf(hdr+26,"%ld",&nsamples);
    while(*iexp--)
        samprate = samprate*10;
    if (NULL!=(keyptr=findlbl(hdr+NOTESTART,RN))){
        sscanf(keyptr,"RN %[0-9a-zA-Z]",sfield);
        if(stricmp(sfield,fname))
            return(RNBAD); /* not equal if non zero */
        }
    else
        return(RNNO);
    if (NULL!=(keyptr=findlbl(hdr+NOTESTART,SR))){
        sscanf(keyptr,"SR %lu",&hsrate);
        if(hsrate!=samprate)
            return(SRBAD); /* sample rates don't agree */
        }
    else
        return(SRNO);
    if (NULL!=(keyptr=findlbl(hdr+NOTESTART,CS))){
        sscanf(keyptr,"CS %le",&cutsize);
        cutsize-=(double)nsamples/samprate;
        if(cutsize > 0.1 || cutsize < -0.1) /* error gtr 0.1 sec */
            return(CSBAD); /* cut size off */
        }
    else
        return(CSNO);
    if (nsamples != (fsiz-HEADBYTES)/2)
        return(SIZBAD);
    return(0);
    } /* headbad() */
```

```
main(int argc, char *argv[]){
    FILE *fstat,*fcopy,*fdelete,*fcheck;
    char header[HEADBYTES+1]; char fnamerec[9],fname[80];
    int fin, ifiles=0, iwrong=0, found, headval;
    struct ffblk filerec;
    long flength;
    strcpy(fname,argv[1]); /* must terminate with a backslash */
    strcat(fname,"*.kay");
    if(NULL==(fcopy=fopen("kcopy.bat","w")))
        errexit("error opening kcopy file");
    if(NULL==(fdelete=fopen("kdelete.bat","w")))
        errexit("error opening kdelete file");
    if(NULL==(fcheck=fopen("kdiagnos.bat","w")))
        errexit("error opening kdiagnos file");
    fprintf(fcheck,"echo off\n");
    found = !findfirst(fname,&filerec,0);
    while(found){
        ifiles++;
        header[HEADBYTES]=0; /* null terminate the header string */
        strcpy(fname,argv[1]); /* must terminate with a backslash */
        strcat(fname,filerec.ff_name);
        flength = fopeninp(fname,&fin)
        lseek(fin,0,SEEK_SET);
        headval=read(fin,header,HEADBYTES);
        fnamerec[8]=0;
        memcpy(fnamerec,filerec.ff_name,8);
        if(headval=headbad(fnamerec,header,flength)){
/*          printf("%s %02d/%02d/%02d %02d:%02d ",
                                filerec.ff_name,
                                (filerec.ff_fdate&0x1ff)>>5,
                                filerec.ff_fdate&0x1f,
                                (filerec.ff_fdate>>9)+80,
                                filerec.ff_ftime>>11,
                                (filerec.ff_ftime&0x3ff)>>5);*/
            iwrong++;
            switch (headval){
                case RNNO :
                    fprintf(fcheck,"echo RN not found\npause\n");
                    break;
                case SRNO :
                    fprintf(fcheck,"echo SR not found\npause\n");
                    break;
                case CSNO :
                    fprintf(fcheck,"echo CS not found\npause\n");
                    break;
                case RNBAD :
                    fprintf(fcheck,"echo RN does not agree with filename\npause\n");
                    break;
                case SRBAD :
                    fprintf(fcheck,"echo SR does not agree with kay value\npause\n");
                    break;
                case CSBAD :
                    fprintf(fcheck,"echo CS off by more than 0.1 sec\npause\n");
                    break;
                case SIZBAD :
                    fprintf(fcheck,"echo file length != number of samples\npause\n");
                    } /* switch */
            fprintf(fcheck,"headedit %s\n",fname);
            } /* if headval */
        else if (headval=datscan(fin,fstat)){
/*          printf("%s %02d/%02d/%02d %02d:%02d ",
                                filerec.ff_name,
                                (filerec.ff_fdate&0x1ff)>>5,
                                filerec.ff_fdate&0x1f,
```

```
                                (filerec.ff_fdate>>9)+80,
                                filerec.ff_ftime>>11,
                                (filerec.ff_ftime&0x3ff)>>5);*/
            if(-1==headval)
                fprintf(fcheck,"echo DATA Overrun\npause\n");
            else
                fprintf(fcheck,"echo DATA Underflow\npause\n");
            fprintf(fcheck,"msg %s\n",fname);
            iwrong++;
            }
        else{
            fprintf(fcopy,"copy %s %%1\n",fname);
            fprintf(fdelete,"del %s\n",fname);
            }
        close(fin);
        found = !findnext(&filerec);
        } /* while found */
    printf("%d files found, %d had problems\n",ifiles,iwrong);
    fcloseall();
    return(0);
    } /* main */
```

```
#define DRDEFAULT 27.0
#define DSDEFAULT 0.0
#define UNLOAD
#define SCRFILE
#include "whtimio.c"
#include "nsgbls.h"
#include "dac.c"
/*extern unsigned _stklen=0x2000;*/
/* externals used to permit access from external source files */
char filename[NAMESIZE], pathname[NAMESIZE],
        kolors[VVSCALE], header[HEADWORDS<<1];
/* globals for video */
int vidpage, vscale=VVSCALE, hscale=HSCALE, batgmode=VBSTGMODE,
    botline=VBOTLINE, ovidmode;
int idat[MAXFSIZE], drsaturate;
int xfsize;
float scaler=1;
long sampfreq;
double dynrange;
/*
global floats: avgpow assigned only in getscale
rdat assigned, referenced many places, colscale assigned in
pstest near getscale, maxpow a diagnostic used (potentially) many places
*/
float rdat[MAXFSIZE+2], avgpow[(MAXFSIZE+2)>>1], colscale, maxpow;
/*
TICMARKS
plots ticmarks to aid interpretation during drawing
*/
void ticmarks(long hc, int vmin, int fplot){
    char colr;
    double ticval;
    int i,ticline,ticexp;
/*
approx 5 horizontal (time) tics, rounded to 1 significant digit
*/
    ticexp=floor(ticval=log10((double)hc*HSCALE/(5*sampfreq)));
    ticval=floor(pow(10.0,ticval-ticexp)+0.5);
    for(colr=3,i=1;
        HSCALE > (ticline=
            i*ticval*sampfreq*pow(10.0,(double)ticexp)/(double)hc);i++){
        drawVLn(ticline,vmin,vscale,&colr,0);
        gotoxy((int)((long)ticline*80/HSCALE),3);
        itoa((int)ticval*i,kolors,10);
        writString(kolors,5,vidpage);
        }
    gotoxy(40-8,2);
    writString("seconds x10",6,vidpage);
    gotoxy(43,1);
    itoa(ticexp,kolors,10);
    writString(kolors,6,vidpage);
    if (fplot){
/*
approx 5 vertical (freq) tics; assumes never <1 Hz per tic
*/
        ticexp=floor(ticval=log10(sampfreq/10.0));
        ticval=floor(pow(10.0,ticval-ticexp)+0.5)*pow(10.0,(double)ticexp);
        for(colr=3,i=1;
            256 > (ticline=2*i*ticval*256/sampfreq-1);i++){ /* xfsize */
            drawHLn(vscale-ticline,0,HSCALE-1,colr);
            gotoxy(70,(vscale-ticline)/16);
            ltoa((long)ticval*i,kolors,10);
            strcat(kolors,"Hz");
            writString(kolors,5,vidpage);
```

```
            }
        }
    } /* ticmarks */
/*
LINEPLOT()
divides the time domain sequence in ifl into hinc sized cells
starting at sample firstd (==byte 2*firstd); hinc precomputed
and draws a vertical line extending from the maximum value to the
minimum value. the vertical scale of the plot remains set by the
maximum found in the file.
*/
int lineplot(int ifl, long firstd, long hinc,
                int mnval, int mxval, int fqplot,int xnoise)
    {
    char colr;
    int vvmin, vvmax, vmax, vmin, vidmin, vidmax, i;
    long skptween, skpafter;
    int ibuf, nblkavg, resid;
    int hline, noeof, *iptr;
    float vmult, vadd, *fptr;
    /*cls((char)0x70);*/
    resid=xfsize; /* initialize partial screen blank marker 9-26-91 */
    setgraphics(bstgmode);
    vidmin=16; vidmax=(fqplot ? vscale - SPEKDISP : vscale);
    ticmarks(hinc,vidmin,fqplot);
    vmult=(float)(vidmin - vidmax)/(mxval-mnval);
    vadd=vidmax - vmult*mnval;
    lseek(ifl,((long)HEADWORDS+firstd) << 1,SEEK_SET);
    if(16 < (nblkavg=hinc/256)) /* xfsize */
        nblkavg=16;
    if(nblkavg){
        skptween=(hinc-(long)256*nblkavg)/nblkavg; /* xfsize */
        gain_set(gain_set(0.0)/(double)nblkavg); /* adjust gain: incoherent */
        }
    else{ /* !nblkavg */
        skptween=0;
        idatread(ifl,256,(long)0,idat); /* xfsixe */
        }
    skpafter=hinc%(256+skptween); /* xfaize */
/*
if small hinc causes data overlap, get initial block
subsequently, the old data is shifted right and new
data is filled in on the left
*/
    for(noeof=1, hline=0; hscale>hline; hline++){
        vmax=-(vmin=MAXINT);
        fptr=rdat; iptr=idat; /* used in both halves of if hinc>xfsize */
        if(nblkavg){
            memset(rdat,0,2+xfsize<<2);
            for(ibuf=0; ibuf++ < nblkavg;){
                noeof=idatread(ifl,xfsize,skptween,idat);
                fminmax(idat, noeof, &vvmin, &vvmax);
                vmin=min(vmin, vvmin); vmax=max(vmax, vvmax);
                if (fqplot)
                    for(i=0; i++ < noeof; *fptr++ +=*iptr++);
                fptr=rdat; iptr=idat; /* reset pointers */
                } /* done with whole blocks */
            if(fqplot){
                fldemean(rdat,xfsize);
                makePspec(rdat,xfsize);
                if(xnoise)
                    normPspec(rdat,avgpow,xfsize);
                colrPspec(rdat,kolors,256,colscale); /* xfsize */
                drawVLn(hline,vscale-256,vscale-1,kolors,1); /* xfsize */
```

```
                }
            noeof=idatread(ifl,(int)skpafter,(long)0,idat);
            fminmax(idat, noeof, &vvmin, &vvmax);
            vmin=min(vmin, vvmin); vmax=max(vmax, vvmax);
            } /* if hinc > xfsize */
        else{ /* overlapping segments */
            if (resid>0){
                fminmax(idat, hinc, &vmin, &vmax);
                if (fqplot){
                    imean2fp(idat,rdat,xfsize);
                    makePspec(rdat,xfsize);
                    if(xnoise)
                        normPspec(rdat,avgpow,256); /* xfsize */
                    colrPspec(rdat, kolors,256,colscale); /* xfsize */
                    drawVLn(hline,vscale-256,vscale-1,kolors,1); /* xfsize */
                    }
                memcpy(idat,idat+hinc,(256-hinc)<<1); /* xfsize */
                noeof=idatread(ifl,hinc,(long)0,idat+256-hinc); /* xfsize */
                resid-=hinc-noeof;
                }
            else{
                *kolors=0;
                drawVLn(hline,1,vscale,kolors,0);
                }
            } /* if hinc > xfsize...else */
        if (resid>0){
            vmin=floor(0.5 + vadd + vmult*vmin);
            vmax=floor(0.5 + vadd + vmult*vmax);
            vmin=min(vmin,vidmax);
            vmax=max(vmax,vidmin);
            colr=0;drawVLn(hline,vidmin,vidmax,&colr,0); /* colr=0 */
            colr=12;drawVLn(hline,vmin,vmax,&colr,0); /* colr=12 */
        }
        if(bioskey(1)){
            if (nblkavg)
                gain_set(gain_set(0.0)*(double)nblkavg); /* reset gain */
                return(-1);
            }
        } /* for hline */
    return(0);
    } /* function lineplot */
/*
VCURSE
*fln, *lln mark the begin and end cursor positions
fdisp and ldisp mark the begin and end data on the display
*lln and ldisp mark the first data beyond the cursor/display
imov can be positive or negative, and it has been multiplied by hscale already
*/
void vcurse(long *fln, long *lln, long fdisp, long ldisp,
                long imov, unsigned lcurs)
    {
    if (lcurs){
        if (ldisp < (*lln +=imov))
            *lln=ldisp;
        else if (*lln < *fln)
            *lln=*fln;
        }
    else
        if (fdisp > (*fln +=imov))
            *fln=fdisp;
        else if (*lln < *fln)
            *fln=*lln;
    }
/*
```

```
HCURSE
*fbar, *lbar mark the begin and end frequency bars
imov can be positive or negative
upbar is nonzero for upper bar
*/
void hcurse(int *ub, int *db, int imov, int upbar)
    {
    if (upbar){
        if (SPEKDISP-1 < (*ub +=imov))
            *ub=SPEKDISP-1;
        else if (*ub < *db)
            *ub=*db;
            }
    else
        if (2 > (*db +=imov))
            *db=2;
        else if (*ub < *db)
            *db=*ub;
    }
/*
TIMCAT
timcat adds a minutes/seconds ascii string to global string kolors
*/
void timcat(long jcurse){
    int i,j,k;
    char ims[5], *cp;
    ltoa(jcurse/(60*sampfreq),kolors+strlen(kolors),10);
    strcat(kolors,":");
    ltoa((jcurse/sampfreq)%60,kolors+strlen(kolors),10);
    strcat(kolors,".");
    ltoa((jcurse*(long)1000/sampfreq)%(long)1000,ims,10);
    if(0<(j=3-(k=strlen(ims)))){
        for(cp=ims+k-1, i=0; k>i++; *(cp+j)=*cp,cp--);
        for(i=0, cp=ims; i++<j; *cp++='0');
        *(ims+3)=0; /* terminate with null */
        }
    strcpy(kolors+strlen(kolors),ims);
    strcat(kolors,"s");
    }
/*
CRSWRIT
saves repetitive code in getcurses()
*/
void crswrit(long icurse, int row, int lstcurse, int ip, long hnc){
    int j, smin, smax;
    gotoxy(20,row);
    strcpy(kolors,"n=");
    ltoa(icurse,kolors+strlen(kolors),10);
    strcat(kolors,", t=");
    timcat(icurse);
    strcat(kolors,", a=");
    smax=-(smin=MAXINT);
    lseek(ip,(HEADWORDS+icurse)<<1,SEEK_SET);
    j=min(hnc,idatread(ip,xfalse,(long)0,idat));
    fminmax(idat,j,&smin,&smax);
    itoa(smin,kolors+strlen(kolors),10);
    strcat(kolors,"<>");
    itoa(smax,kolors+strlen(kolors),10);
    strcat(kolors," pb=");
    ltoa(sampfreq*scaler,kolors+strlen(kolors),10);
    strcat(kolors,"Hz");
    if (row)
        writString(kolors,16+CURSCOL+lstcurse,vidpage);
    else
```

```
        writString(kolors,24+CURSCOL-lstcurse,vidpage);
    } /* crswrit() */
/*
GETCURSES()
assumes that the screen is already in graphics mode, and that the time
domain plot has already been drawn.
defines region of interest using begin and end markers (vertical lines);
firstd is the starting data sample, lastd is the ending data sample
hinc is the horizontal data increment
*/
int getcurses(long *fvline, long *lvline, int *ubar, int *dbar,
                    long *fcurs, long *lcurs, long hinc,
                    int freqplot, int xnoise, int ifp)
    {
    char colr;
    int iwait;
    unsigned int done, ich, j, lastcurse, lastbar;
    lastcurse=8; /* start with the end marker */
    lastbar=4;
    colr=CURSCOL+24;
    if (hscale > (j=(*lcurs - *fvline)/hinc))
        drawVLn(j,16,vscale-1,&colr,0); /* write the line */
    colr=CURSCOL+16;
    if (0 < (j=(*fcurs - *fvline)/hinc))
        drawVLn(j,16,vscale-1,&colr,0); /* write the line */
    if (freqplot){
        colr=FCURSCOL+20;
        drawHLn(vscale-*ubar ,0,639,colr);
        drawHLn(vscale-*ubar+1,0,639,colr);
        colr=FCURSCOL+16;
        drawHLn(vscale-*dbar ,0,639,colr);
        drawHLn(vscale-*dbar+1,0,639,colr);
        }
    if (*fcurs < *fvline)
        *fcurs=*fvline;
    if (*lcurs >*lvline)
        *lcurs=*lvline;
    done=done; /* zero done */
    while(!done){ /* loop; active curse is highlighted as LIGHTRED */
        while(0xff & (ich=bioskey(0))) /*loop until kbd sends null char*/
            if(ich==EXITPROG)return(EXITPROG);
        iwait=1;/* borrowing iwait for speed shift */
        switch(ich){
            case 0x7400 : /* ctrl-right arrow */
            case 0x4d36 : /* shift-right arrow, move marker right 5 */
                iwait=-iwait;
            case 0x7300 : /* ctrl-left arrow */
            case 0x4b34 : /* shift-left arrow, move marker left 5 */
                iwait *=8;
            case 0x4b00 : /* left arrow, so move marker left */
                iwait=-iwait;
            case 0x4d00 : /* right arrow, so move marker right */
                colr=CURSCOL+24;
                if ((0 < (j=((lastcurse?*lcurs:*fcurs) - *fvline)/hinc))&&(hscale > j))
                    drawVLn(j,16,vscale-1,&colr,0); /* xors old line */
                vcurse(fcurs,lcurs,*fvline,*lvline,(long)iwait*hinc,lastcurse);
                if ((0 < (j=((lastcurse?*lcurs:*fcurs) - *fvline)/hinc))&&(hscale > j))
                    drawVLn(j,16,vscale-1,&colr,0); /* xors old line */
                break;
            case 0x5000 : /* down arrow, move bar down one bin */
                iwait=-iwait;
            case 0x4800 : /* up arrow, move bar up one bin */
                if(freqplot){
                    j=vscale-(lastbar?*ubar:*dbar);
```

```
                    drawHLn(j++,0,639,FCURSCOL+20);
                    drawHLn(j ,0,639,FCURSCOL+20);
                    hcurse(ubar, dbar, 2*iwait, lastbar);
                    j=vscale-(lastbar?*ubar:*dbar);
                    drawHLn(j++,0,639,FCURSCOL+20);
                    drawHLn(j ,0,639,FCURSCOL+20);
                    }
                break;
            case 0x4700 : /* keypad del key, toggle lastcurse */
                colr=16+(CURSCOL^(CURSCOL+8));
                if ((0 < (j=(*lcurs - *fvline)/hinc))&&(hscale > j))
                    drawVLn(j,16,vscale-1,&colr,0); /* xors old line */
                if ((0 < (j=(*fcurs - *fvline)/hinc))&&(hscale > j))
                    drawVLn(j,16,vscale-1,&colr,0); /* xors old line */
                lastcurse ^=8;
                if(freqplot){
                    colr=16+(FCURSCOL^(FCURSCOL+4));
                    drawHLn(vscale-*ubar ,0,639,colr);
                    drawHLn(vscale-*ubar+1,0,639,colr);
                    drawHLn(vscale-*dbar ,0,639,colr);
                    drawHLn(vscale-*dbar+1,0,639,colr);
                    lastbar ^=4;
                    }
                break;
            case 0x6800 : /* Alt-F1, increase freq by a factor of 2 */
                scaler *= 2;
                if (sampfreq*scaler > 81920)
                {
                    printf("\x7");
                    scaler /= 2;
                }
                break;
            case 0x6900 : /* Alt-F2, decrease freq by a factor of 2 */
                scaler /= 2;
                if (sampfreq*scaler < 640)
                {
                    printf("\x7");
                    scaler *= 2;
                }
                break;
            case 0x1900 : /* alt-p, playback data between cursors */
                start_dac(ifp,fcurs,lcurs,sampfreq,scaler);
                break;
            case AAPLUS :
            case AAMINUS :
            case DRPLUS : /* */
            case DRMINUS : /* */
            case FDEFAULT : /* alt-d, restore sonagram defaults */
            case XNOISE : /* alt-n, toggle noise suppression */
            case WRTFILE : /* save data within cursors in new file */
            case PSHGSTACK : /* control page down, redisplay with new limits */
            case POPGSTACK : /* control page up, pop previous cursors */
            case FREQDISP : /* alt-f, draw sonagram */
            case EXITPROG : /* escape, done moving markers, exit */
            case FFTNCRSE : /* alt-F5, increase fft-size by a factor of 2 */
            case FFTDCRSE : /* alt-F6, decrease fft-size by a factor of 2 */
                done=1; /* terminates outer while loop, action taken */
                                /* in the psteat() while loop */
        } /* switch ich */
    iwait=0;
    if (!done)
        while(!biosakey(1))
            if(5000<=iwait++){
                clrline(botline);
```

```
                    clrline(0);
                    strcpy(kolors,filename);
                    writString(kolors,5,vidpage);
                    crswrit(*fcurs,0,lastcurse,ifp,hinc); /* front/upper cursor */
                    if (xnoise)
                        writString(" noise referenced", 5, vidpage);
                    gotoxy(0,botline);
                    if (freqplot){
                        ltoa((*ubar>>1)*sampfreq/SPEKDISP,kolors,10);
                        writString(kolors,FCURSCOL+16+lastbar,vidpage);
                        writChar('-',5,vidpage);cursorinc(1,vidpage);
                        ltoa((*dbar>>1)*sampfreq/SPEKDISP,kolors,10);
                        writString(kolors,FCURSCOL+20-lastbar,vidpage);
                        writString("=",5,vidpage);
                        ltoa(((*ubar>>1)-(*dbar>>1))*sampfreq/SPEKDISP,kolors,10);
                        writString(kolors,FCURSCOL+18,vidpage);
                        writString("Hz",5,vidpage);
                        }
                    crswrit(*lcurs,botline,lastcurse,ifp,hinc);
                    strcpy(kolors,", dt=");
                    timcat(*lcurs-*fcurs);
                    writString(kolors,20+CURSCOL,vidpage);
                    break;
                    } /* if j++==1000 */
            } /* while (!done) */
        *f.line=*fcurs;
        *lvline=*lcurs;
        return(ich);
        } /* function getcurse */
/*
IWRTFILE
*/
int wrtfile(int ifile, long fdisp, long ldisp){
        int i, nrd, itext, *iptr;
        long isamp, *mh;
        char fname[40], *cptr;
        char mathead[50];
        FILE *ofl;
        /*cls((char)0x70);*/
        setgraphics(batgmode);
        gotoxy(0,0);
        writString("output file name: ",0xf,vidpage);
        strset(fname,0);
        while (0==*fname) /* readString returns NULL if escaped */
            readString(fname,0);
        if (itext=(NULL !=strstr(fname,".tex")))
            ofl=fopen(fname,"w");
        else
            ofl=fopen (fname, "wb");
        if (ofl==NULL){
            writString("destination File opening did not succeed... ",0x0f,vidpage);
            return(-1);
            }
        lseek(ifile,(HEADWORDS+fdisp)<<1,SEEK_SET); /* same for all formats */
        if (itext){ /* writing in ascii */
            for(isamp=fdisp; isamp<ldisp; isamp+=nrd){
                nrd=idatread(ifile,nrd=min(xfaize,ldisp-isamp),(long)0,iptr=idat);
                for(i=0; i++<nrd; ){
                    itoa(*iptr++,cptr=fname,10);cptr--;
                    while(*++cptr) /* my fwritestring function */
                        putc((int)*cptr,ofl);
                    putc('\n',ofl);
                    } /* for i */
                } /* for isamp */
```

```
        } /* if itext */
    else if (NULL !=strstr(fname,".mat")){ /* writing matlab */
        mh=mathead;
        *mh++=40;
        *mh++=1;
        *mh++=ldisp-fdisp;
        *mh++=0;
        ultoa(sampfreq,kolors,10);
        *mh=strlen(kolors)+2;
        *(mathead+20)='s';
        strcpy(mathead+21,kolors);
        fwrite(mathead,1,21+strlen(mathead+20),ofl);
        for(isamp=fdisp; isamp<ldisp; isamp+=nrd){
            nrd=idatread(ifile,nrd=min(xfaize,ldisp-isamp),(long)0,idat);
            fwrite(idat,2,nrd,ofl);
            } /* for isamp */
        } /* else */
    else { /* writing kay, with header */
        ltoa(ldisp-fdisp,fname,10);
        if (6 > strlen(fname))
            for(i=0, cptr=header+26; i++ < 6-strlen(fname); *cptr++='0');
        strcpy(cptr,fname);
        fwrite(header,2,HEADWORDS,ofl);
        for(isamp=fdisp; isamp<ldisp; isamp+=nrd){
            nrd=idatread(ifile,nrd=min(xfaize,ldisp-isamp),(long)0,idat);
            fwrite(idat,2,nrd,ofl);
            } /* for isamp */
        } /* else */
    fclose(ofl);
    return(0);
    } /* wrtfile */
/*
PRMCHANGE()
changes GLOBALS dynrange, drsaturate, colscale, avgpow
space saving function for psteet
*/
int prmchange(unsigned iv,int *sg,int *kn){
    int rescale;
    rescale=0;
    switch(iv){
        case FDEFAULT : /* restore defaults */
            *sg=1;*kn=0;
            dynrange=DRDEFAULT;
            drsaturate=DSDEFAULT;
            rescale=-1;
            break;
        case XNOISE :
            *kn=!*kn;
            rescale=-1;
            break;
        case FREQDISP : *sg=!*sg; break;
        case DRPLUS :

gain_set(gain_set(0.0)*pow(10.0L,(MAXCOLOR*3.0L)/((MAXCOLOR-1)*20.0L)));
            colscale *=dynrange/(dynrange+3);
            dynrange +=3.0;
            break;
        case DRMINUS :
            if (dynrange > 3.0){

gain_set(gain_set(0.0)*pow(10.0L,(MAXCOLOR*3.0)/((1-MAXCOLOR)*20.0L)));
                colscale *=dynrange/(dynrange-3.0);
                dynrange -=3.0;
                }
```

```
            break;
        case AAPLUS :
            gain_set(gain_set(0.0)*pow(10.0L,1.0L/20.0L));
            drsaturate++;
            break;
        case AAMINUS :
            gain_set(gain_set(0.0)*pow(10.0L,-1.0L/20.0L));
            drsaturate--;
            break;
        case FFTNCRSE :
            xfsize *= 2;
            if (xfsize > 256)
            {
                printf("\x7"); /* beep to indicate limit */
                xfsize = 256;
            }
            break;
        case FFTDCRSE :
            xfsize /= 2;
            if (xfsize < 64)
            {
                printf("\x7"); /* beep to indicate limit */
                xfsize = 64;
            }
            break;
        } /* switch iv */
    return(rescale);
    } /* prmchange() */
/* SCALESET
uses the results of getscale to determine scaling for power spectral
displays
*/
float scaleset(float maxpow,int kn){
    maxpow -=drsaturate;
    if(kn)
        colscale=MAXCOLOR/maxpow;
    else{
        gain_set(pow(10.0,MAXCOLOR*dynrange/(20.0*(MAXCOLOR-1))-maxpow/20.0));
        colscale=(double)(MAXCOLOR-1.0)/dynrange;
        maxpow +=(double)20.0*log(gain_set(0.0))/log((double)10.0);
        }
    return(maxpow);
    } /* scaleset */
/*
psfest
*/
void psfest(void)
    {
    int maxval, minval, ival, istack, ifl;
    int killnoise=0, sonag=1, *sfmant, *sfexp, upbar=SPEKDISP, downbar=2;
/*
8/6/91? CHECK ON UP AND DOWNBAR INDICES AND FREQUENCY TRANSLATIONS
XFSTACK NOT YET IMPLEMENTED
CHANGES IN XFSIZE WILL AFFECT MANY OTHER PARAMS, INCL. AVGPOW
*/
    int mnvstack[STKSZ], mxvstack[STKSZ], xfstack[MAXFSIZE];
    long fdisplay, ldisplay, numsamp, bbinsize, isamp, fcurse, lcurse;
    long fdstack[STKSZ], ldstack[STKSZ], bbinstak[STKSZ];
    float gainstak[STKSZ], clscstak[STKSZ], glob_mean, ss_noise;
    vidpage=getPage();
    ovidmode=getMode(&ival);
/*
filename (global) filled in MAIN (naig)
            or SCRFILE:
```

```
a macro that gets the filename from the screen (iplot)
a null macro when compiled as part of NSIG
*/
    SCRFILE
    fcurse=fdisplay=0;
    setgraphics(batgmode);
    lcurse=ldisplay=numsamp=(fopeninp(filename,&ifl)-512)>>1;
    hbinsize=numsamp/hscale+1;
    writString("SIG 10/03/91 \n",7,vidpage);
    xfsize=fft_size(SPEKDISP);
    polar_mode(MODE_DEFLT);
    gain_set(1.0);
    lseek(ifl,0,SEEK_SET);
    if (HEADWORDS<<1 !=(ival=read(ifl,header,HEADWORDS<<1)))
        errexit("complete header not found");
    sfmant=header+122; sfexp=header+120;sampfreq="sfmant;
    for(ival=0;ival<"sfexp;sampfreq"=10,ival++);
    *avgpow=0.0; /* zero in avgpow signals first entry */
    prmchange(FDEFAULT,&fdisplay,&killnoise);
    maxpow=getscale(ifl, fdisplay, ldisplay, xfsize, idat, rdat,
                                &minval, &maxval, killnoise, hscale, avgpow,
                                &glob_mean, &ss_noise);
    maxpow=scaleset(maxpow,killnoise);
/*
now draw the lines
*/
    ival=istack=0;
/*
enters the graphic display here
starts out with the rear graphics cursors selected
left and right arrows move the cursor in single steps (hscale points)
ctrl key speed up this movement 5x
home key selects front cursor, end selects rear cursor
pgdn expands such that selected region fills the screen
pgup pops out the previously selected region
six levels of stack are provided
esc terminates the program
*/
    while (EXITPROG !=ival){
        while(bioskey(1))
            bioskey(0);
        while (lineplot(ifl, fdisplay, hbinsize, minval, maxval,
                        sonag, killnoise))
            while(bioskey(1)){
                if(EXITPROG==(ival=bioskey(0))){
                    writString("escaped from screen drawing",0x4,vidpage);
                    setMode(ovidmode);
                    return;
                    }
                else if(prmchange(ival,&sonag,&killnoise)){
                    maxpow=getscale(ifl, fdisplay, ldisplay, xfsize, idat,
                        rdat,     &minval, &maxval, killnoise, hscale, avgpow,
                        &glob_mean, &ss_noise);
                    maxpow=scaleset(maxpow,killnoise);
                    } /* if prmchange...else */
                } /* while bioskey(1) */
/* Make title */
        gotoxy(0,0);
        strcpy(kolors,filename);
        strcat(kolors," n=");
        ltoa(numsamp,kolors+strlen(kolors),10);
        strcat(kolors," min=");
        itoa(minval,kolors+strlen(kolors),10);
        strcat(kolors," max=");
```

```
        itoa(maxval,kolors+strlen(kolors),10);
        strcat(kolors," sf=");
        ltoa(sampfreq,kolors+strlen(kolors),10);
        strcat(kolors,"Hz dr=");
        itoa((int)dynrange,kolors+strlen(kolors),10);
        strcat(kolors," aa=");
        itoa((int)drsaturate,kolors+strlen(kolors),10);
        strcat(kolors," fft=");
        itoa(xfsize,kolors+strlen(kolors),10);
        writString(kolors,TITLCOL,vidpage);
        if (killnoise)writString(" NR", 9, vidpage);
/*
reads the data between the cursors (using fdisplay and hbinsize)
switches to graphics mode and clears the screen
*/
        fdstack[istack]=fdisplay; /* save current locations */
        ldstack[istack]=ldisplay;
        if (PSHGSTACK==(ival=getcurses(&fdisplay,&ldisplay, &upbar, &downbar,
                                                    &fcurse, &lcurse,
                                                    hbinsize,sonag,killnoise,ifl))){
/*
fd and ldstack store current screen edges (f and ldisplay have cursors)
must save scaling values before incrementing the pointer
*/
            if ((STKSZ-1 > istack) &&
                ((fdisplay !=fdstack[istack]) ||
                (ldisplay !=ldstack[istack]))){
                mnvstack[istack]=minval;
                mxvstack[istack]=maxval;
                hbinstak[istack]=hbinsize;
                gainstak[istack]=gain_set(0.0);
                clscstak[istack++]=colscale;/* increment the stack pointer */
                }
            hbinsize=(ldisplay-fdisplay)/hscale;
            if((ldisplay-fdisplay)%hscale)
                hbinsize++;
            ldisplay=fdisplay+hscale*hbinsize;
            maxpow=getscale(ifl, fdisplay, ldisplay, xfsize, idat, rdat,
                &minval, &maxval, killnoise, hscale, avgpow,
                &glob_mean, &ss_noise);
            maxpow=scaleset(maxpow,killnoise);
            } /* if PSHGSTACK */
        else if (POPGSTACK==ival){
            if (0 < istack){
                minval=mnvstack[-istack];
                maxval=mxvstack[istack];
                hbinsize=hbinstak[istack];
                gain_set(gainstak[istack]);
                colscale=clscstak[istack];
                }
            fdisplay=fdstack[istack];
            ldisplay=ldstack[istack];
            } /* if POPGSTACK */
        else{
            if (WRTFILE==ival)
                if(wrtfile(ifl,fdisplay,ldisplay)){
                    gotoxy(0,0);
                    writString("error writing file",0xf,vidpage);
                    }
/*
getcurses returns the cursor positions in fdisplay and ldisplay
must recover true display positions from the stack if no change
desired, as in WRTFILE and FRQDISP
*/
```

```
                fdisplay=fdstack[istack];
                ldisplay=ldstack[istack];
                if(prmchange(ival,&sonag,&killnoise)){
                    getscale(ifl, fdisplay, ldisplay, xfaise, idat, rdat,
                        &minval, &maxval, killnoise, hscale, avgpow,
                        &glob_mean, &ss_noise);
                    maxpow=scaleset(maxpow,killnoise);
                    }
                }
            if (numsamp < (ldisplay=fdisplay + (long)hscale*bbinsize))
                ldisplay=numsamp;
            } /* while EXITPROG !=ival */
        close(ifl);
        setMode(ovidmode);
        return;
        } /* psiest */
main(argc,argv)
int argc; char *argv[];
{
        if (1 == argc)
            errexit("usage: nsig <filename> dr aa [e|v]");
        strcpy(filename,argv[1]);
        if ('e' == (argv[4][0] | 0x20)){ /* binary or converts to lower case */
            vscale = EVSCALE;
            bstgmode = EBSTGMODE;
            botline = EBOTLINE;
        } /* defaults are VVSCALE, VBSTGMODE, VBOTLINE */
        vscale-=16;
        if (!(dynrange = atof(argv[2])))
            dynrange = DRDEFAULT;
        if (!(dnaturate = atoi(argv[3])))
            dnaturate=DSDEFAULT;
        psiest();
        return(0);
}
```

```
/*
dac.c
This function reads and sends the kay file data to a Canetics board model
PC-DMA. The D/A capabilities of the Canetics board are used to 'play' the
kay file while the time series and time-frequency distribution are displayed
by SIG.EXE.
The following functions were taken from the Model PC-DMA User Manual,
Canetics, Inc.
memreq()
init_dma_ad()
init_dma_da()
init_d,a_da_cont()
init_board()
kill_timer()
init_timer_da()
The following function was taken from Graphics Programming in C, Roger T.
Stevens, M&T Publishing, Inc., Redwood City, California, 1988.
plot()
*/
#include <lib180.h>
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <alloc.h>
#include <conio.h>
#include <fcntl.h>
#include <stat.h>
#include <string.h>
#include <io.h>
#include <fcntl.h>
#include <math.h>
#define CONTROLA /* add+1*4 */ 15876
#define CONTROLB /* add+2*4 */ 15880
#define OFFSET_ADD /* add+10*4 */ 15912
#define RESET_ADD /* add+11*4 */ 15916
#define DMAMODE_ADD     11
#define DMA_MASK_ADD     10
#define CLRPTR_ADD 12
#define AD_TIMER_MODE 122          /* ad is timer #1 in this mode */
#define DA_TIMER_MODE 52          /* and da is timer #0 */
#define TIMER_AD 15925
#define TIMER_DA 15924
#define TIMER_ADD 15924
#define TIMER_MODE_ADD 15927
#define STA     15904
#define STD     15908
#define TOGGLE_DAC_ADDRESS 15888
#define UH_SEG(p) (( FP_SEG( p ) + ((FP_OFF( p ) ) >> 4) ) & 0xF000)
#define UH_OFF(p)
((int)((((FP_SEG(p)+((FP_OFF(p))>>4))&0x0FFF)<<4)+(FP_OFF(p)&0x000F)))
typedef struct settings *psets;
struct settings sets1;
psets sets;
char curfilname[30];
unsigned long data_size;
int mem_allc=0;
int far *memreq(void) /* Allocate two pages of memory and choose */
{ /* one complete page */
    int tempseg;
    void far *death_to_nonbelievers;
    death_to_nonbelievers = farmalloc(0x20000L);
    if (death_to_nonbelievers == NULL)
    {
        printf("Coreleft = %ld Need %ld\n\r", coreleft(), 0x20000L );
```

```
        putchar(1);
        exit(1);
    }
    tempseg = UH_SEG(death_to_nonbelievers);
    if (UH_OFF(death_to_nonbelievers) != 0)
        tempseg += 0x1000;
    return (int far *)MK_FP(tempseg, 0);
}
void init_dma_ad(unsigned count, unsigned seg)
{
outp(DMA_MASK_ADD, 0x05);
outp(DMAMODE_ADD, 0x45);
outp(CLRPTR_ADD, 0);
outp(2, 0);
outp(2, 0);
outp(3, count & 0x00FF);
outp(3, count >> 8);
outp(130, seg);
outp(131, seg);
outp(DMA_MASK_ADD, 1);
}
void init_dma_da(unsigned count, unsigned seg)
{
outp(DMA_MASK_ADD, 0x07);
outp(DMAMODE_ADD, 0x4B);
outp(CLRPTR_ADD, 0);
outp(6, 0);
outp(6, 0);
outp(7, count & 0x00FF);
outp(7, count >> 256);
outp(130, seg);
outp(131, seg);
outp(DMA_MASK_ADD, 3);
}
void init_dma_da_cont(unsigned seg)
{
outp(DMA_MASK_ADD, 0x07);
outp(DMAMODE_ADD, 0x5B);
outp(CLRPTR_ADD, 0);
outp(6, 0);
outp(6, 0);
    outp(7, 0xFF);
    outp(7, 0xFF);
    outp(130, seg);
    outp(131, seg);
    outp(DMA_MASK_ADD, 0x03);
}
void init_board(struct settings *sets)
{
    outp(RESET_ADD, 0);
    outp(OFFSET_ADD, sets->offset);
    outp(CONTROLA,209); /* PC-DMA Control register A: 11010001 */
    outp(CONTROLB,154); /* PC-DMA Control register B: 10011010 */
    outp(RESET_ADD, 0);
}
void kill_timer(struct settings *sets)
{
outp(RESET_ADD, 0);
outp(CONTROLB, 15 + 16*sets->gainbits + 128);
outp(RESET_ADD, 0);
if (sets->dac2flag == 2)
outp(TOGGLE_DAC_ADDRESS, 0);
outp(TIMER_MODE_ADD, 50);
outp(TIMER_MODE_ADD, 114);
```

```
outp(TIMER_MODE_ADD, 178);
    init_dma_da(0, sets->seg);
    init_dma_ad(0, sets->seg);
}
void init_timer_da(unsigned spaces)
{
outp(TIMER_MODE_ADD, DA_TIMER_MODE);
outp(TIMER_DA, spaces & 0x00FF);
outp(TIMER_DA, spaces >> 8 );
}
void plot (int x, int y, int color)
{
    unsigned int offset;
    int dummy, mask;
    char far *mem_address;
    offset = (long)y*80L + ((long)x/8L);
    mask = 0x80 >> (x%8);
    _ES = 0xA000;
    _BX = offset;
    _CX = color;
    _AX = mask;
    asm MOV AH,AL
    asm MOV AL,08
    asm MOV DX,03CEH
    asm OUT DX,AX
    asm MOV AX,0FF02H
    asm MOV DL,0C4H
    asm OUT DX,AX
    asm OR ES:[BX],CH
    asm MOV BYTE PTR ES:[BX],00
    asm MOV AH,CL
    asm OUT DX,AX
    asm MOV BYTE PTR ES:[BX],0FFH
    asm MOV AH,0FFH
    asm OUT DX,AX
    asm MOV DL,0CEH
    asm MOV AX,0003
    asm OUT DX,AX
    asm MOV AX,0FF08H
    asm OUT DX,AX
}
void wtfile_cont(psets sets, int curfil)
{
    unsigned ii, jj, tempcount, ckpoint=0, last, same, buffer[0x4000], loc=0;
    unsigned far *block, far *uptr;
    unsigned near *nptr;
    unsigned long chunk, max_pt, point;
    double curs_inc;
    block = MK_FP(sets->seg << 12, 0); /* Pointer to allocated memory */
    kill_timer(sets);
    max_pt = data_size;
    curs_inc = (double)640/(double)data_size;
    plot(0,215,15);
    if (data_size < 0x10000) /* If < one page, read in all data and */
    { /* then cycle to PC-DMA */
        plot(0,215,0);
        point = ceil(curs_inc*(double)(max_pt-data_size));
        plot(point,215,15);
        uptr = block;
        while(data_size > 0x4000)
        {
            read(curfil, buffer, 0x4000);
            data_size -= 0x4000;
            loc += 0x4000;
```

```
            ckpoint += 64;
            nptr = buffer;
            while (FP_OFF(uptr) < loc)
                *uptr++ = *nptr++ <<= 4;
        }
        plot(point,215,0);
        point = ceil(curs_inc*(double)(max_pt-data_size));
        plot(point,215,15);
        read(curfil, buffer, data_size);
        loc += data_size;
        ckpoint += 64;
        if (ckpoint == 319)
            ckpoint = 63;
        nptr = buffer;
        while (FP_OFF(uptr) < loc)
            *uptr++ = *nptr++ <<= 4;
        same = *uptr;
        while (FP_OFF(uptr) < loc - data_size + 0x4000)
            *uptr++ = same;
        init_board(sets);
        init_dma_da_cont(sets->seg);
        init_timer_da(sets->waits);
        outp(CLRPTR_ADD, 0);
        inp(6);
        tempcount = inp(6);
        while (tempcount < ckpoint)
        {
            inp(6);
            tempcount = inp(6);
        }
    }
    if (data_size > 0x10000) /* If > one page, fill one page and then */
    { /* fill quarter page as DMA passes proceeds */
        uptr = block;
        for (ii=0;ii<4;ii++)
        {
            read(curfil, buffer, 0x4000);
            data_size -= 0x4000;
            nptr = buffer;
            loc += 0x4000;
            if (loc == 0)
                loc = 0xfffe;
            while (FP_OFF(uptr) < loc)
                *uptr++ = *nptr++ <<= 4;
            *uptr = *nptr <<= 4;
        }
        uptr++;
        init_board(sets);
        init_dma_da_cont(sets->seg);
        init_timer_da(sets->waits);
        outp(CLRPTR_ADD, 0);
        plot(0,215,0);
        point = ceil(curs_inc*(double)(max_pt-data_size-0x10000));
        plot(point,215,15);
        inp(6);
        tempcount = inp(6);
        while (tempcount < 63)
        {
            inp(6);
            tempcount = inp(6);
        }
        for (ii=1;ii<sets->num_of_pag;ii++)
        {
            ckpoint = 127;
```

```
        chunk = 0x4000;
        for (jj=1; jj<=4; jj++)
        {
            read(curfil, buffer, 0x4000);
            data_size -= 0x4000;
            plot(point,215,0);
            point = ceil(curs_inc*(double)(max_pt-data_size-0x10000));
            plot(point,215,15);
            nptr = buffer;
            while (FP_OFF(uptr) < chunk)
                *uptr++ = *nptr++ <<= 4;
            *uptr =*nptr <<= 4; /* ch */
            inp(6);
            tempcount = inp(6);
            while ( tempcount < ckpoint )
            {
                inp(6);
                tempcount = inp(6);
            }
            chunk += 0x4000;
            if (chunk == 0x10000)
                chunk -= 2;
            ckpoint += 64;
            if (ckpoint == 319)
                ckpoint = 63;
        }
        uptr++;
    }
    last = ceil((double)data_size/0x4000);
    ckpoint = 127;
    chunk = 0x4000;
    for (jj=1; jj<last; jj++)
    {
        read(curfil, buffer, 0x4000);
        data_size -= 0x4000;
        plot(point,215,0);
        point = ceil(curs_inc*(double)(max_pt-data_size-0x10000));
        plot(point,215,15);
        nptr = buffer;
        while (FP_OFF(uptr) < chunk)
            *uptr++ = *nptr++ <<= 4;
        *uptr = *nptr <<= 4;
        inp(6);
        tempcount = inp(6);
        while ( tempcount < ckpoint )
        {
            inp(6);
            tempcount = inp(6);
        }
        chunk += 0x4000;
        if (chunk == 0x10000)
            chunk -= 2;
        ckpoint += 64;
        if (ckpoint == 319)
            ckpoint = 63;
    }
    plot(point,215,0);
    point = ceil(curs_inc*(double)(max_pt-0x10000));
    plot(point,215,15);
    read(curfil, buffer, data_size);
    nptr = buffer;
    while (FP_OFF(uptr) < chunk - 0x4000 + data_size)
        *uptr++ = *nptr++ <<= 4;
    *uptr++ = *nptr++ <<= 4;
```

```
        inp(6);
        tempcount = inp(6);
        while ( tempcount < ckpoint ) /* Watch DMA and wait until data */
        { /* has all been sent to board */
            inp(6);
            tempcount = inp(6);
        }
        plot(point,215,0);
        point = ceil(curs_inc*(double)(max_pt-0x4000));
        plot(point,215,15);
        ckpoint += 64;
        if (ckpoint == 319)
            ckpoint = 63;
        inp(6);
        tempcount = inp(6);
        while ( tempcount < 255 )
        {
            inp(6);
            tempcount = inp(6);
        }
        plot(point,215,0);
        point = ceil(curs_inc*(double)(max_pt-0x8000));
        plot(point,215,15);
        if (ckpoint == 63)
            ckpoint = 255;
        else
            ckpoint -= 63;
        inp(6);
        tempcount = inp(6);
        while ( tempcount < ckpoint )
        {
            inp(6);
            tempcount = inp(6);
        }
        plot(point,215,0);
        point = ceil(curs_inc*(double)(max_pt-0xc000));
        plot(point,215,15);
        if (ckpoint == 63)
            ckpoint = 255;
        else
            ckpoint -= 63;
        inp(6);
        tempcount = inp(6);
        while ( tempcount < ckpoint )
        {
            inp(6);
            tempcount = inp(6);
        }
    }
    kill_timer(sets);
    plot(point,215,0);
}
void start_dac(int ifile, long *begin, long *end, long sample_rate,
                        float scal)
{
    int far *M_Ptr;
    int *amant,*sexp;
    char header[512];
    outp(RESET_ADD,0xda); /* Detect Canetics board */
    if (inp(RESET_ADD) == 0) /* 0 indicates board is present */
    {
        sets = &sets1; /* Get the structure 'sets' variables */
        if (!mem_allc) /* Allocate memory only once */
        {
```

```
            M_Ptr = memreq();
            sets->seg = FP_SEG( M_Ptr ) >> 12;
            mem_allc=1;
        }
        if (*begin == 0) /* Move to start in kay file */
            lseek(ifile,512,SEEK_SET);
        else
            lseek(ifile,*begin*2+512,SEEK_SET);
        data_size = 2*(*end - *begin);
        sample_rate *= scal;
        sets->waits = floor(1e6/sample_rate + 0.5);
        sets->num_of_pag = floor((double)data_size/0x10000);
        sets->datasize = 1;
        sets->mline = 0;
        sets->runstop = 0;
        sets->gainbits = 1;
        sets->inmode = 2;
        sets->offset = 128;
        sets->filterstate = 1;
        sets->usingxmux = 0;
        sets->dac2flag = 1;
        init_board(sets);
        kill_timer(sets);
        wtfile_cons(sets,ifile);
    }
}
```

```
/* byte mask always 0xff, color in di, mask in si, jlabel replaces noxor */
/* bx: memory location; dx, ax: general purpose registers */
/*
DRAWhLN
ASSUMES SMALL MEMORY MODEL
draws a single pixel vertical line, located at horizontal position x
vertical bounds of the line are y1 and y2
xor function can be selected with color=color+16, overwrite otherwise
cchange = increment for color pointer
*/
#pragma inline
void drawHLn(int y, int x1, int x2, char color){
/*
exchange y1 and y2 if y1 larger than y2
*/
    if (x1 > x2){
        asm mov bx,x2
        asm mov ax,x1
        asm mov x2,ax
        asm mov x1,bx
        }
/*
any use of egaqix requires bracketing by the aix port
instructions bracketing the for loop below
see Dwyer et. al. micro/systems july88.20-34
*/
/* setting up ega for write mode 2 */
    outportb(0x3ce,5);
    outportb(0x3cf,2);
/* set cl = x1 mod 8 */
    asm mov ax,x1
    asm mov cl,al
/* set ch = x1 div 8: ranges from 0-80 */
    asm sar ax,1
    asm sar ax,1
    asm sar ax,1
    asm mov ch,al
/* bx = 80 * y1 */
    asm mov ax,y
    asm sal ax,1
    asm sal ax,1
    asm sal ax,1
    asm sal ax,1
    asm mov bx,ax
    asm sal ax,1
    asm sal ax,1
    asm add bx,ax
/* bx = 80 * y + x/8 */
    asm xor ax,ax
    asm mov al,ch
    asm add bx,ax
/* bx = address offset */
/* es = ega ram starting address */
    asm mov ax,0xa000
    asm mov es,ax
/* set di = color */
    asm mov di, color
    asm and di,0x1f;
/* setup complete, remaining items vary */
/* now plot first partial byte, seroing the initial bits, put mask in si*/
    asm mov al,0xff
    asm and cl, 0x7
    asm js iskip
    asm shl al,cl
```

```
    asm shr al,cl
iskip:
    asm xor ah,ah
    asm mov si,ax
/*
select bit mask register
output bit mask
back to select register
function select register
reset bits 3 and 4
check for xor function
set bits 3 and 4
*/
    asm mov dx,0x3ce
    asm mov al,0x8
    asm out dx,al
    asm inc dx
    asm mov ax,si
    asm out dx,al
    asm dec dx
    asm mov al,0x3
    asm out dx,al
    asm inc dx
    asm xor al,al
    asm test di,0x10
    asm jz initp
    asm mov al,0x18
initp:
    asm out dx,al
/* mask and function selected */
    asm mov ah,es:[bx]
    asm mov ax,di
    asm mov es:[bx],al
/* now prepare for all whole byte plots */
/* compute x2 div 8, and subtract x1 div 8 */
    asm mov ax,x2
    asm sar ax,1
    asm sar ax,1
    asm sar ax,1
    asm sub al,ch
    asm dec ax
    asm jz last
/*
save loop count in cx
select bit mask register
output bit mask
*/
    asm mov cx,ax
    asm mov dx,0x3ce
    asm mov al,0x8
    asm out dx,al
    asm inc dx
    asm mov ax,0xff
    asm out dx,al
/* loop */
ploop:
    asm inc bx
    asm mov ah,es:[bx]
    asm mov ax,di
    asm mov es:[bx],al
    asm loop ploop
/* now prepare last partial byte */
/* set cl = x2 mod 8 */
last:
```

```
        asm mov ax,x2
        asm mov cl,0x7
        asm and cl,al
        asm inc bx
        asm mov al,0xff
        asm xor cl,0x7
        asm jz lskip
        asm shr al,cl
        asm shl al,cl
lskip:
        asm mov si,ax
/*
select bit mask register
output bit mask
back to select register
function select register
reset bits 3 and 4
check for xor function
set bits 3 and 4
*/
        asm mov dx,0x3ce
        asm mov al,0x8
        asm out dx,al
        asm inc dx
        asm mov ax,si
        asm out dx,al
        asm mov ah,es:[bx]
        asm mov ax,di
        asm mov es:[bx],al
        outportb(0x3ce,5);
        outportb(0x3cf,0);
        outportb(0x3ce,8);
        outportb(0x3cf,0xff);
/* returning ega to write mode 0 */
        } /* drawvln */
```

```
#define MAXBOOST -15.0
#include "nagbls.h"
/*
this file contains all of the fft related code, including any functions
that call the fft routines
it also contains a few static global variables that are particular to
the fft and related subroutines
*/
/* MUST CHANGE THE MAXBOOST CONSTANT AND ITS PURPOSE */
extern void far pascal
    rfft(float far *dat, int far *exp, int far *norm, float far *scl);
extern void far pascal
    polar(float far *idata, float far *odat, long far *ndatsize, int far *cmode);
extern void far pascal
    hamm(float far *rdata, float far *sdat, int far *exp);
/*
GLOBALS used by MicroWay 87 routines
*/
static int normalize, pow2; /* used by rfft */
static int mode;/* 6 for naig, iplot; 1 for kaystats */
static long datsize; /* used by polar */
static float gain; /* used by rfft */
/*
these functions initialize and/or report GLOBAL variables
used in MicroWay routines
GLOBALS MUST NOT BE MODIFIED ANYWHERE ELSE IN FFTFUNCS, AND
THESE PUBLIC FUNCTIONS ARE THE ONLY ACCESS FOR OTHER SOURCE FILES
*/
int fft_size(int xf){
    normalize = 2; /* rfft */
    if(xf){
        datsize = (xf>>1); /* polar */
        pow2 = 0; /* rfft */
        while(xf>>++(pow2));
        (pow2)--;
        }
    return((int)datsize<<1);
    } /* fft_size */
int polar_mode(int p_m){
    if(p_m)
        mode=p_m;
    return(mode);
    }
float gain_set(double gn){
    if(gn)
        gain=(float)gn;
    return(gain);
    }
/*
FLDEMEAN
computes and subtracts out the mean of a float array
new FOR loop requires pointer arithmatic and logical comparisons
*/
void fldemean(float *fdata, int isize){
    float *fptr; double mean;
    for(fptr=fdata+isize, mean=0.0; --fptr>=fdata; mean+=*fptr);
    mean /= isize;
    for(fptr=fdata+isize; --fptr>=fdata; *fptr-=mean);
    } /* fldemean */
/*
IMEAN2FP
demeans integer data in idat, places it in rdat
*/
void imean2fp(int *idat, float *rdat, int isize){
```

```
    int *iptr;
    long imean;
    float *fptr;
    double dmean;
    for(imean=0,iptr=idat+isize; --iptr>=idat; imean+=*iptr);
    dmean = (double)imean/(double)isize;
    for(fptr=rdat+isize,iptr=idat+isize; fptr>rdat; *--fptr=*--iptr-dmean);
    }
/*
MAKEpSPEC
calls canned hamming window, canned fft routine for real data
converts complex results to mag, phase and returns in rdat
GLOBALS INITIALIZED IN INITFFT()
*/
void makePspec(float *rdat, int isize){
    hamm(rdat, rdat, &pow2);
    *(rdat+(isize>>1)) += 0.001; /* prevents underflow in log power */
    rfft(rdat, &pow2, &normalise, &gain);
    polar(rdat, rdat, &datasize, &mode);
    } /* makePspec() */
/*
NORMpSPEC
subtracts the power spectrum values in avgp[] from rdat[]
note that rdat has PS values in even entries only
            avgp has PS values in every entry
*/
void normPspec(float *rdat, float *avgp, int isize){
    float *faig;
    for(faig=rdat+isize, avgp+=(isize>>1);
        faig>rdat; *(faig-=2)-=*--avgp);
    }
/*
COLRpSPEC
converts power spectrum values in rdat to integer color values in kolrs
relying on pointer arithmatic and logical comparisons for iptr, kolrs
this version allows fft sizes other than 256
*/
void colrPspec(float *rdat, char *kolrs, int isize, float colscale){
    char c;
    int *iptr, ncells, k, j;
    float *fptr;
    ncells=SPEKDISP/isize;
/* iptr is placed SPEKDISP bytes beyond the beginning of kolrs */
    for(fptr=rdat+isize, iptr=kolrs+SPEKDISP; iptr>kolrs; ){
        k = (int)(colscale* *(fptr-=2));
        k = min(MAXCOLOR,max(0,k));
        for(j=0; j++<ncells; *--iptr=(k|(k<<8)));
        }
    } /* colrPspec() */
/*
SDIFFCMP
*/
int sdiffcmp(const void *rec1, const void *rec2){
    float *f1, *f2;
    f1 = rec1; f2 = rec2;
    if(*f1 > *f2)
        return(1);
    else if (*f1 < *f2)
        return(-1);
    else
        return(0);
    }
/*
GETSCALE
```

```
*/
float getscale(int ifp, long fdisp, long ldisp,      int xfaize,
                        int idat[], float rdat[], int *imin, int *imax,
                        int xnoise, int hscale, float avgp[],
                        float *g_mean, float *ss_noise){
    typedef struct {float ss; int num;} ssblk;
    ssblk blk[HSCALE];
    int *iptr, iold, i, diff, jmax, jmin, nblocks, nsamp, imaxss;
    int firstblok, lastblok;
    float sumsq, maxss, *fptr, *dptr, mxpow, global_mean;
    double ssn_mean;
    long lskip, fplace;
    gain_set(1.0);
    memset(blk,0,sizeof(ssblk)*HSCALE);
    lseek(ifp,(fdisp+HEADWORDS)<<1,SEEK_SET);
    *imax = -(*imin = MAXINT);
    for(global_mean = maxss = imaxss = i = 0,
        nblocks=min(hscale,(ldisp-fdisp)/xfaize),
        lskip = max(0,(ldisp - fdisp)/hscale-xfaize);
        i < nblocks; i++){
        fplace=tell(ifp);
        nsamp = idatread(ifp,xfaize,lskip,idat);
/* file handler, #items, bytes to skip after read, *destination, ret #ints */
        for(sumsq=0.0,global_mean+=iold=jmax=jmin=*(iptr=idat+nsamp-1);
            --iptr>=idat; ){
            global_mean+=*iptr;
            diff = *iptr-iold;
            sumsq += diff*diff;
            if(*iptr > jmax)
                jmax = *iptr;
            else if (*iptr < jmin)
                jmin = *iptr;
            iold=*iptr;
            }
        if(*imax < jmax)
            *imax = jmax;
        if(*imin > jmin)
            *imin = jmin;
        blk[blk[i].num = i].ss = sumsq;
        if(maxss<sumsq){ /* needed for rescaling colors during soom */
            maxss=sumsq; /* when full qsort is not performed */+
            imaxss=i;
            }
        } /* for i 0 to nblocks-1 */
    *g_mean=global_mean/((double)nblocks*xfaize);
    lskip += xfaize; /* lskip becomes read interval */
/* polar_mode(straight amplitude) */
    if (*avgp==0.0){ /* only done once for the file */
        qsort(blk,nblocks,sizeof(ssblk),sdiffcmp);
        memset(avgp,0,(1+(xfaize>>1))*sizeof(float));
        ssn_mean=0.0;
        firstblok = nblocks/10;
        if(!(lastblok=nblocks/5))
            lastblok++;/* 10-20% */
        for(i=firstblok; i<lastblok; /* i++ in loop body */){
            ssn_mean+=blk[i].ss;
            lseek(ifp,(HEADWORDS+blk[i++].num*lskip)<<1,SEEK_SET);
            nsamp = idatread(ifp,xfaize,(long)0,idat);
            imean2fp(idat,rdat,xfaize);
            makePspec(rdat,xfaize);
            for(dptr=avgp+xfaize/2, fptr = rdat+xfaize; (fptr-=2)>=rdat;
                *--dptr += *fptr);
            } /* for i= firstblok to lastblok */
        *ss_noise=ssn_mean/(double)(lastblok-firstblok);
```

```
/*————————compute scaling for power spectra————————*/
/*
THIS NEEDS WORK: BETTER MAXBOOST VALUE
if polar_mode reset, need to compute 20ln(*dptr/sumsq)
*/
        for(dptr=avgp+xfsize/2, sumsq = lastblok-firstblok;
            --dptr>=avgp; )
            if(MAXBOOST > (*dptr /= sumsq))
                *dptr=MAXBOOST;
        } /* if (*avgp==0.0) */
/*————————goto maximum block————————————————*/
/* polar_mode(20ln(amplitude)) */
    if ((long)-1==lseek(ifp,(long)(fdisp+imaxss*lskip+HEADWORDS)<<1,SEEK_SET))
        errexit("error in seeking maximum sumsquares block");
    idatread(ifp,(int)xfsize,(long)0,idat);
    imean2fp(idat,rdat,xfsize);
    makePspec(rdat,xfsize);
    if (xnoise)
        normPspec(rdat,avgp,xfsize);
    for(mxpow=*(fptr=rdat), i=1;
        i++ < (xfsize>>1);){
        if(*(fptr+=2) > mxpow)
            mxpow = *fptr;
        }
    return(mxpow);
    } /* getscale */
```

```
struct settings
{ unsigned mline;
unsigned runstop;
unsigned gainbits;
unsigned filterstate;
unsigned datasize;
unsigned inmode;
unsigned seg;
unsigned offset;
unsigned waits;
     unsigned long num_of_con;
unsigned dac2flag;
unsigned num_of_pag;
unsigned usingxmux;
};
void init_dma_ad(unsigned count, unsigned seg);
void init_dma_da(unsigned count, unsigned seg);
void init_dma_ad_cont(unsigned seg);
void init_dma_da_cont(unsigned seg);
void init_board(struct settings *sets);
void kill_timer(struct settings *sets);
void init_timer_da(unsigned spaces);
void init_timer_ad(unsigned spaces);
void writesingle(struct settings *sets, unsigned msb, unsigned lsb);
void readfile(struct settings *sets);
void writefile(struct settings *sets);
void readfile_cont(struct settings *sets, char *filename);
void writefile_cont(struct settings *sets, char *filename);
int status(int print);
unsigned readsingle(struct settings *sets);
#define G_VERSION "2.20"
```

```
/*
DRAWVLN 11/24/89
ASSUMES SMALL MEMORY MODEL
draws a single pixel vertical line, located at horizontal position x
vertical bounds of the line are y1 and y2
xor function can be selected with color=color+16, overwrite otherwise
!this function is presumed to be selected for the entire line!
cchange = increment for color pointer
*/
#pragma inline
void drawVLn(int x, int y1, int y2, char *color, int cchange){
/*
exchange y1 and y2 if y1 larger than y2
*/
    if (y1 > y2){
        asm mov bx,y2
        asm mov ax,y1
        asm mov y2,ax
        asm mov y1,bx
        }
/*
any use of egaqix requires bracketing by the six port
instructions bracketing the for loop below
egaqix taken from micro/systems july88:20-34
*/
/* setting up ega for write mode 2 */
    outportb(0x3ce,5);
    outportb(0x3cf,2);
/* set up function register */
    asm mov dx,0x3ce
    asm mov al,0x3
    asm out dx,al
/* function select register */
    asm inc dx
/* reset bits 3 and 4 */
    asm xor al,al
/* set si color string pointer, set di = current color */
    asm mov si,color
    asm mov di, [si]
/* check for xor function */
    asm test di,0x10
    asm js noxor
/* set bits 3 and 4 */
    asm mov al,0x18
noxor:
    asm out dx,al
/* bx = 80 * y2 */
    asm mov ax,y2
    asm sal ax,1
    asm sal ax,1
    asm sal ax,1
    asm sal ax,1
    asm mov bx,ax
    asm sal ax,1
    asm sal ax,1
    asm add bx,ax
/* bx = 80 * y2 + x/8 */
/* set cl = x mod 8 */
    asm mov ax,x
    asm mov cx,7
    asm and cx,ax
/* compute x div 8 (ranges from 0-80), add to bx */
    asm sar ax,1
    asm sar ax,1
```

```
    asm sar ax,1
    asm add bx,ax
/* bx = address offset, color already in di */
    asm mov ax,0xa000
    asm mov es,ax
/* es = ega ram starting address */
/* select bit mask register */
    asm mov dx,0x3ce
    asm mov al,0x8
    asm out dx,al
/* output bit mask */
    asm inc dx
    asm mov al,0x80
    asm ror al,cl
    asm out dx,al
/*
DO NOT ASSIGN VALUES TO CX, SI, DI IN THE LOOP
*/
    if(cchange)
        for (; y2- >= y1; ){ /* egaqix(x,y,color); */
            asm mov ah,es:[bx]
            asm mov ax,di
            asm mov es:[bx],al
            asm sub bx,80
            asm inc si
            asm mov di, [si]
            } /* for y2 */
    else
        for (; y2- >= y1; ){ /* egaqix(x,y,color); */
            asm mov ah,es:[bx]
            asm mov ax,di
            asm mov es:[bx],al
            asm sub bx,80
            } /* for y2 */
    outportb(0x3ce,5);
    outportb(0x3cf,0);
    outportb(0x3ce,8);
    outportb(0x3cf,0xff);
/* returning ega to write mode 0 */
    } /* drawvln */
```

```
/*
SPEKDISP: number of vertical pixels used to display power spectra
MAXFSIZE: maximum fft size
HEADWORDS: size of key header in 2-byte words
MAXCOLOR: the highest color number used for power spectral display
?BSTGMODE: video mode number for the best graphics mode
?VSCALE: number of vertical pixels in gmode
?BOTLINE: last text line number in gmode
STKSZ: the size of the FILO stacks for zoom/restore
HSCALE: the number of horizontal pixels (same for ega, vga)
*/
#define SPEKDISP 256
#define MAXFSIZE     256
#define HEADWORDS 256
#define MAXINT 0X7FFF
#define MAXCOLOR 15
#define VBSTGMODE 0X12
#define VVSCALE 480
#define EBSTGMODE 0X10
#define EVSCALE 350
#define VBOTLINE 29
#define EBOTLINE 24
#define STKSZ 6
#define FDEFAULT 0x2000 /* Alt-d */
#define XNOISE 0x3100 /* Alt-n */
#define FREQDISP 0x2100 /* Alt-f */
#define DRPLUS 0x5200 /* insert */
#define DRMINUS 0x5300 /* delete */
#define AAPLUS 0x7700 /* ctrl-home */
#define AAMINUS 0x7500 /* ctrl-end */
#define POPGSTACK 0x4900 /* PgUp */
#define EXITPROG 0x11b /* Esc */
#define PSHGSTACK 0x5100 /* PgDn */
#define WRTFILE 0x7200 /* Ctrl-PrintScrn */
#define FFTNCRSE 0x6c00 /* Alt-F3 */
#define FFTDCRSE 0x6d00 /* Alt-F4 */
#define ENTER 13
#define BACKSPACE 8
#define NAMESIZE 40
#define CURSCOL 6
#define TITLCOL 9
#define FCURSCOL 8
#define HSCALE 640
#define MODE_DEFLT 6
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <string.h>
```

```
/* prototypes generated from functions in kaystuff\fftfuncs.c */
int fft_size(int xf);
int polar_mode(int p_m);
float gain_set(double gn);
void fldemean(float *fdata, int isize);
void imean2fp(int idat[], float rdat[], int isize);
void makePspec(float rdat[], int isize);
void normPspec(float rdat[], float *avgp, int isize);
void colrPspec(float rdat[], char kolrs[], int isize, float colscale);
int sdiffcmp(const void *rec1, const void *rec2);
float getscale(int ifp, long fdisp, long ldisp, int xfsize, int idat[], float rdat[], int *imin, int *imax,
int xnoise, int hscale, float avgp[], float *g_mean, float *ss_noise);
/* prototypes generated from functions in kaystuff\drawhln.c */
void drawHLn(int y, int x1, int x2, char color);
/* prototypes generated from functions in kaystuff\ndrawvln.c */
void drawVLn(int x, int y1, int y2, char *color, int cchange);
/* prototypes generated from functions in kaystuff\comsig.c */
void ticmarks(long hc, int vmin, int fplot);
int lineplot(int ifl, long firstd, long hinc, int mnval, int mxval, int fqplot,int xnoise);
void vcurse(long *fln, long *lln, long fdisp, long ldisp, long imov, unsigned lcurs);
void hcurse(int *ub, int *db, int imov, int upbar);
void timcat(long jcurse);
void crswrit(long icurse, int row, int lstcurse, int ip, long hnc);
int getcurses(long *fvline, long *lvline, int *ubar, int *dbar, long *fcurs, long *lcurs, long hinc,
int freqplot, int xnoise, int ifp);
int wrtfile(int ifile, long fdisp, long ldisp);
int prmchange(unsigned iv,int *sg,int *kn);
float scaleset(float maxpow,int kn);
void pstest(void);
/* prototypes generated from functions in kaystuff\whtimio.c */
void errexit(char *msg);
int idatread(int ip, int xsize, long lskip, void *dat);
long fopeninp(char *fname, int *ifile);
void fminmax(int idat[], int nelem, int *vmin, int *vmax);
/* prototypes generated from functions in kaystuff\plotutls.c */
int readChar(int page, char *attr);
int getkey();
void gotoxy(int column, int row);
int getPage(void);
void getxy(int *column, int *row, int page);
void writChar(char ch, int color, int page);
int getMode(int *ncols);
void cursorinc(int spaces,int page);
void crlf(int page);
void writString(char *str, int color, int page);
void setMode(int mode);
void cls(char colors);
void readString(char *sptr, int count);
void setpalette(int palette, int color);
void setgraphics(int gmode);
void clrline(int line);
```

```
#include "nsig.pro"
#include <dos.h>
#define BACKSPACE 8
#define ENTER 13
/*
READcHAR
Reads a Character from the Screen
*/
int readChar(int page, char *attr)
{
    union REGS reg;
    reg.h.ah = 8;
    reg.h.bh = page;
    int86 (0x10,&reg,&reg);
    *attr = reg.h.ah;
    return (reg.h.al);
    }
/*
GETKEY()
uses ROM-BIOS service to get a keycode from buffer, or wait for keyboard
*/
int getkey()
{
    union REGS reg;
    reg.h.ah = 0;
    return(int86(0x16,&reg,&reg)); /* turboC int86() returns AX */
    }
/*
GOTOXY()
Moves Cursor to Specified x,y Position and Page 3
*/
void gotoxy(int column, int row)
{
    union REGS reg;
    reg.h.ah = 2;
    reg.h.bh = 0;
    reg.h.dh = row;
    reg.h.dl = column;
    int86 (0x10,&reg,&reg);
}
/*
GETpAGE()
Returns Active Page Number
*/
int getPage(void)
{
    union REGS reg;
    reg.h.ah =0x0F;
    int86(0x10,&reg,&reg);
    return (reg.h.bh);
}
/*
GETXY()
Gets the Current Cursor Position
*/
void getxy(int *column, int *row, int page)
{
    union REGS reg;
    reg.h.ah = 3;
    reg.h.bh = page;
    int86 (0x10,&reg,&reg);
    *row = reg.h.dh;
    *column = reg.h.dl;
}
```

```
/*
WRITcHAR
Writes a Character to the Screen with Specified Attribute
*/
void writChar(char ch, int color, int page)
{
    union REGS reg;
    reg.h.ah = 9;
    reg.h.al = ch;
    reg.h.bl = color;
    reg.h.bh = page;
    reg.x.cx = 1;
    int86 (0x10,&reg,&reg);
    }
/*
GETmODE()
Returns Current Video Mode
*/
int getMode(int *ncols)
{
    union REGS reg;
    reg.h.ah = 0x0F;
    int86 (0x10,&reg,&reg);
    *ncols = reg.h.ah;
    return reg.h.al;
}
/*
CURSORINC
moves the cursor right one space, wraps lines and page
*/
void cursorinc(spaces,page)
    int spaces,page;
{
    int col, row, width, length;
    if ((0x11 == (length = getMode (&width))) || (0x12 == length))
        length = 30;
    else
        length = 25;
    getxy(&col,&row,page);
    if ((col += spaces) >= width){
        col = 0;
        row++;
        }
    else if (col <0)
        col = 0;
    if (row >= length)
        row = 0;
    gotoxy(col, row);
    }
/*
CRLF
moves the cursor to the left margin, down one line
*/
void crlf(page)
    int page;
{
    int col, row;
    getxy(&col,&row,page);
    gotoxy(0,row+1);
    }
/*
WRITsTRING
Writes a String to the Screen with Specified Attribute
can write faster if cursorinc functions are placed in the
```

```
function initialization, and col and row are updated within
the routine.
*/
void writString(char *str, int color, int page)
{
    char cval; int i = 0;
    while (cval=str[i]){
        if ('\n'==cval){
            crlf(page);
            }
        else{
            writChar(cval, color, page);
            cursorinc(1,page);
            }
        i++;
        }
    }
/*
SETmODE()
Sets Video Mode
*/
void setMode(int mode)
{
    union REGS reg;
    reg.h.ah = 0;
    reg.h.al = mode;
    int86 (0x10,&reg,&reg);
}
/*
CLS() = Clears the Screen
char colors = (B7)blink, (B6-B4)rgb foreground, (B3)foreground intensity,
                (B2-B0)rgb backround
backround sets the fill color used
colors has a different meaning under CGA
taken from Graphics Programming in C: RT Stevens 1988
*/
void cls(char colors)
{
    union REGS reg;
    int columns,mode;
    mode = getMode(&columns);
    if (columns == 80)
    {
        if ((mode == 0x11) || (mode == 0x12))
            reg.x.dx = 0x1D4F;
        else
            reg.x.dx = 0x184F;
        reg.h.bh = colors;
    }
    else
    {
        reg.x.dx = 0x1828;
        switch (colors)
        {
            case 1:     reg.h.bh = 0x55;
                break;
            case 2:     reg.h.bh = 0xAA;
                break;
            case 3:     reg.h.bh = 0xFF;
                break;
            default:
                reg.h.bh = 0;
                break;
        }
```

```
    }
    reg.x.ax = 0x0600;
    reg.x.cx = 0;
    int86(0x10,&reg,&reg);
    gotoxy(0,0);
}
/*
READsTRING
Reads a string from the keyboard, terminated with a \n
*/
#define ESCAPE 0x1b
void readString(char *sptr, int count)
{
    int vidpage;
    vidpage=getPage();
    while(ENTER != (*sptr++=getkey() & 0xff))
        if (BACKSPACE == *(sptr-1)){
            if (count){
                cursorinc(-1,vidpage);
                writChar(0x20,0xf,vidpage);
                count-;
                sptr-=2;
                }
            else
                sptr-;
            }
        else if (ESCAPE == *(sptr-1)){
            *(sptr-1-count)=0;
            return;
            }
        else{
            writChar(*(sptr-1),0xf,vidpage);
            cursorinc(1,vidpage);
            count++;
            }
    crlf(vidpage);
    *(sptr-1) = 0;
    }
void setpalette(int palette, int color)
{
    union REGS reg;
    reg.h.ah = 0x10;
    reg.h.al = 0;
    reg.h.bh = color;
    reg.h.bl = palette;
    int86(0x10,&reg,&reg);
}
/*
SETGRAPHICS
*/
void setgraphics(int gmode){
    setMode(gmode);
    setpalette(1,8);setpalette(2,1);setpalette(3,33);setpalette(4,40);
    setpalette(5,5);setpalette(6,45);setpalette(7,61);setpalette(8,47);
    setpalette(9,53);setpalette(10,37);setpalette(11,44);setpalette(12,36);
    setpalette(13,52);setpalette(14,54);setpalette(15,63);
    cls((char)0x70);
    }
/*
CLRLINE
clears a line on the display, using my C-bios routines
*/
void clrline(int line){
    int i, pg;
```

```
    pg = getPage();
    gotoxy(0,line);
    for(i=0;i++<80;writChar(32,1,pg),cursorinc(1,pg));
    gotoxy(0,line);
    }
```

```
#include "naig.pro"
#include "nagbls.h"
extern ovidmode, vidpage;
/*
ERREXIT
*/
void errexit(char *msg){
     fcloseall();
     writString("error: ",5,vidpage);
     writString(msg,5,vidpage);
     delay(2000);
     setMode(ovidmode);
     exit(0);
     }
/*
IDATREAD
*/
int idatread(int ip, int xsize, long lskip, void *dat){
     int bytesread;
     bytesread = read(ip, dat, xsize<<1);
     lseek(ip, lskip<<1, SEEK_CUR);
     return(bytesread>>1);
     }
/*
FOPENINP
*/
long fopeninp(char *fname, int *ifile){
     *ifile = open(fname,O_RDONLY|O_BINARY);
     while (-1 == *ifile){
          writString("Source File opening did not succeed: ",5,vidpage);
          writString(fname,4,vidpage);crlf(vidpage);
          writString("Enter file name for input file: ",15,vidpage);
          strset(fname,0);
          readString(fname,0);
          if (0 == *fname){
               setMode(ovidmode);
               errexit("aborted from file entry");
          }
          UNLOAD
          *ifile = open(fname,O_RDONLY|O_BINARY);
          }
     lseek(*ifile,0,SEEK_END);
     return(tell(*ifile)); /* return number of bytes */
     }
void fminmax(int idat[], int nelem, int *vmin, int *vmax){
     int ival, i, imin, imax, *iptr;
     for(i=0, iptr=idat, imin=imax=*idat; ++i<nelem; )
          if ((ival=*++iptr) > imax)
               imax = ival;
          else if (ival < imin)
               imin = ival;
          *vmax = imax;
          *vmin = imin;
          } /* for j */
```

DOCUMENT LIBRARY

March 11, 1991

Distribution List for Technical Report Exchange

Attn: Stella Sanchez-Wade
Documents Section
Scripps Institution of Oceanography
Library, Mail Code C-075C
La Jolla, CA 92093

Hancock Library of Biology &
Oceanography
Alan Hancock Laboratory
University of Southern California
University Park
Los Angeles, CA 90089-0371

Gifts & Exchanges
Library
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, NS, B2Y 4A2, CANADA

Office of the International
Ice Patrol
c/o Coast Guard R & D Center
Avery Point
Groton, CT 06340

NOAA/EDIS Miami Library Center
4301 Rickenbacker Causeway
Miami, FL 33149

Library
Skidaway Institute of Oceanography
P.O. Box 13687
Savannah, GA 31416

Institute of Geophysics
University of Hawaii
Library Room 252
2525 Correa Road
Honolulu, HI 96822

Marine Resources Information Center
Building E38-320
MIT
Cambridge, MA 02139

Library
Lamont-Doherty Geological
Observatory
Columbia University
Palisades, NY 10964

Library
Serials Department
Oregon State University
Corvallis, OR 97331

Pell Marine Science Library
University of Rhode Island
Narragansett Bay Campus
Narragansett, RI 02882

Working Collection
Texas A&M University
Dept. of Oceanography
College Station, TX 77843

Library
Virginia Institute of Marine Science
Gloucester Point, VA 23062

Fisheries-Oceanography Library
151 Oceanography Teaching Bldg.
University of Washington
Seattle, WA 98195

Library
R.S.M.A.S.
University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

Maury Oceanographic Library
Naval Oceanographic Office
Stennis Space Center
NSTL, MS 39522-5001

Marine Sciences Collection
Mayaguez Campus Library
University of Puerto Rico
Mayaguez, Puerto Rico 00708

Library
Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB
UNITED KINGDOM

The Librarian
CSIRO Marine Laboratories
G.P.O. Box 1538
Hobart, Tasmania
AUSTRALIA 7001

Library
Proudman Oceanographic Laboratory
Bidston Observatory
Birkenhead
Merseyside L43 7 RA
UNITED KINGDOM

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. WHOI-92-11	2.	3. Recipient's Accession No.

4. Title and Subtitle
Software Tools for Acoustic Database Management

5. Report Date
January 1992

6.

7. Author(s) Kurt M. Fristrup, Mary Ann Daher, Terrance J. Howald and William A. Watkins

8. Performing Organization Rept. No.
WHOI-92-11

9. Performing Organization Name and Address

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C) N00014-88-K-0273
(G)

12 Sponsoring Organization Name and Address

Office of Naval Research

13. Type of Report & Period Covered
Technical Report

14.

15. Supplementary Notes

This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-92-11.

16. Abstract (Limit: 200 words)

Digital archiving of bioacoustic data provides both curatorial and scientific benefits. To realize these benefits, key system requirements must be satisfied. This report discusses these requirements, and describes the software tools developed by the WHOI bioacoustic laboratory to maintain and utilize an archive of digitized biological sounds. These tools are written in standard C code, and are designed to run on PC-compatible microcomputers. Both the usage and structure of these programs are described in relation to the SOUND database of marine animal sounds. These tools include software for analog-to-digital conversion, text header maintenance, data verification and interactive spectrographic review. Source code listings are supplied.

17. Document Analysis **a. Descriptors**

marine animal sounds
database management
digital animal sound cuts

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement

Approved for public release; distribution unlimited.

19. Security Class (This Report)
UNCLASSIFIED

20. Security Class (This Page)

21. No. of Pages
94

22. Price

(See ANSI-Z39.18) *See Instructions on Reverse*

OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce